

# ITSO Workshop: VisualAge for Java Version 3

*Ueli Wahli, Claus Schroeder-Hansen, Mikio Yoshino, Jerry Ramirez, Matthias Zieger*



**International Technical Support Organization**

[www.redbooks.ibm.com](http://www.redbooks.ibm.com)





International Technical Support Organization

WS-SW-B402

**ITSO Workshop:  
VisualAge for Java Version 3**

November 1999

**Take Note!**

Before using this information and the product it supports, be sure to read the general information in Appendix B, "Special Notices" on page 481.

**First Edition (November 1999)**

This edition applies to VisualAge for Java Version 3 for use with the Windows NT Operating System. It should also apply to VisualAge for Java Version 3 on OS/2, AIX, and Linux.

Comments may be addressed to:  
IBM Corporation, International Technical Support Organization  
Dept. QXXE Building 80-E2  
650 Harry Road  
San Jose, California 95120-6099

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1999. All rights reserved.

Note to U.S Government Users - Documentation related to restricted rights - Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

---

# Contents

<b>Instructor Course Overview</b> .....	xv
<b>Course Description</b> .....	xvi
<b>Part 1. Lectures</b> .....	19
<b>Unit 1. VA Java Version 3 Introduction and Overview</b> .....	21
Figure 1-2. Objectives .....	22
Figure 1-3. Prerequisites .....	23
Figure 1-4. Agenda .....	24
Figure 1-5. VisualAge for Java Version 3 Professional .....	25
Figure 1-6. VisualAge for Java Version 3 Enterprise .....	26
Figure 1-7. VisualAge for Java Version 3 Enhancements .....	27
Figure 1-8. IDE Enhancements .....	28
Figure 1-9. Visual Composition Enhancements .....	29
Figure 1-10. WebSphere/Servlet Enhancements .....	30
Figure 1-11. Enterprise Enhancements (1) .....	31
Figure 1-12. Enterprise Enhancements (2) .....	32
Figure 1-13. Distributed Debugger Enhancements .....	33
Figure 1-14. Integration Enhancements .....	34
Figure 1-15. Technology Previews .....	35
Figure 1-16. DB2 Issues .....	36
Figure 1-17. Summary .....	37
Figure 1-18. We Integrate the World! .....	38
<b>Unit 2. VA Java Version 3 IDE Enhancements</b> .....	39
Figure 2-2. Objectives .....	40
Figure 2-3. Help: View Reference Help .....	41
Figure 2-4. Help: Open on Selection .....	42
Figure 2-5. Extended Code Assist Feature .....	43
Figure 2-6. Enhanced Filters .....	44
Figure 2-7. Generating Required Methods .....	45
Figure 2-8. Generating Accessors .....	46
Figure 2-9. Enhanced Search/Replace .....	47
Figure 2-10. Maintain Project Resources in VA Java .....	48
Figure 2-11. New SmartGuide for Visual Application .....	49
Figure 2-12. VCE: Layout Management .....	50
Figure 2-13. VCE: Property Settings and Editor .....	51
Figure 2-14. VCE: Swing Improvements .....	52
Figure 2-15. VCE: Code Generation Options .....	53
Figure 2-16. VCE: Quick Form .....	54
Figure 2-17. BeanInfo: Enumeration Values Prompt .....	55
Figure 2-18. BeanInfo: Delete Features .....	56
Figure 2-19. Debugger Enhancements (1) .....	57
Figure 2-20. Debugger Enhancements (2) .....	58
Figure 2-21. Debugger Enhancements (3) .....	59
Figure 2-22. Debugger Enhancements (4) .....	60
Figure 2-23. Customizable Key Bindings .....	61
Figure 2-24. History, Web-Browser like Interface .....	62
Figure 2-25. Summary .....	63

<b>Unit 3. VA Java Version 3 Enhanced Data Access Beans</b>	65
Figure 3-2. Objectives	66
Figure 3-3. Data Access Beans Feature	67
Figure 3-4. Select Bean	68
Figure 3-5. Navigator Bean	69
Figure 3-6. Modify Bean	70
Figure 3-7. Editing the Modify Bean Properties	71
Figure 3-8. Connection Alias Definition	72
Figure 3-9. Using Connection Pools	73
Figure 3-10. SQL Statement Definition	74
Figure 3-11. SQLAssist SmartGuide	75
Figure 3-12. Parameterized SQL Statements	76
Figure 3-13. Using the Modify Bean	77
Figure 3-14. ProcedureCall Bean	78
Figure 3-15. Edit ProcedureCall Bean Properties	79
Figure 3-16. Set up the Database Connection	80
Figure 3-17. Define the SQL Statement	81
Figure 3-18. Define the SQL Statement (2)	82
Figure 3-19. Define the SQL Statement (3)	83
Figure 3-20. Using the ProcedureCall Bean	84
Figure 3-21. Using the ProcedureCall Bean (2)	85
Figure 3-22. Selector Beans	86
Figure 3-23. CellSelector Bean	87
Figure 3-24. CellRangeSelector Bean	88
Figure 3-25. RowSelector Bean	89
Figure 3-26. ColumnSelector Bean	90
Figure 3-27. Using the Selector Beans	91
Figure 3-28. Summary	92
<b>Unit 4. VA Java Version 3 WebSphere Integration</b>	93
Figure 4-2. Objectives	94
Figure 4-3. WebSphere Application Server Overview	95
Figure 4-4. WebSphere Test Environment	96
Figure 4-5. WebSphere Test Env. Directories	97
Figure 4-6. Starting the WebSphere Test Env.	98
Figure 4-7. Servlet Development (1)	99
Figure 4-8. Servlet Development (2)	100
Figure 4-9. Servlet Development (3)	101
Figure 4-10. Servlet Development (4)	102
Figure 4-11. JavaServer Pages Development (1)	103
Figure 4-12. JavaServer Pages Development (2)	104
Figure 4-13. JavaServer Pages Development (3)	105
Figure 4-14. JavaServer Pages Development (4)	106
Figure 4-15. XML Application Testing	107
Figure 4-16. XML Logic Schema	108
Figure 4-17. EJB Application Testing (1)	109
Figure 4-18. EJB Application Testing (2)	110
Figure 4-19. EJB Application Testing (3)	111
Figure 4-20. RMI_IIOP Overview (1)	112
Figure 4-21. RMI_IIOP Overview (2)	113
Figure 4-22. RMI_IIOP Architecture Schema	114
Figure 4-23. Additional Configuration	115
Figure 4-24. Summary	116
<b>Unit 5. VA Java Version 3 Servlet Builder Enhancements</b>	117
Figure 5-2. Objectives	118

Figure 5-3. Overview . . . . .	119
Figure 5-4. Application Flow . . . . .	120
Figure 5-5. New Features . . . . .	121
Figure 5-6. New Features (2) . . . . .	122
Figure 5-7. New Features (3) . . . . .	123
Figure 5-8. Wrapper Connections . . . . .	124
Figure 5-9. Typical Servlet - JSP Model . . . . .	125
Figure 5-10. Servlet Launcher . . . . .	126
Figure 5-11. Servlet Launcher (2) . . . . .	127
Figure 5-12. Summary . . . . .	128
<b>Unit 6. VA Java Version 3 XML Parser for Java . . . . .</b>	<b>129</b>
Figure 6-2. Objectives. . . . .	130
Figure 6-3. XML Overview. . . . .	131
Figure 6-4. IBM XML Parser for Java. . . . .	132
Figure 6-5. XML Parser Example . . . . .	133
Figure 6-6. XML File Example . . . . .	134
Figure 6-7. DTD File Example. . . . .	135
Figure 6-8. Java Handler Code Example . . . . .	136
Figure 6-9. Java Application Code Example. . . . .	137
Figure 6-10. Testing . . . . .	138
Figure 6-11. Reference Sites . . . . .	139
Figure 6-12. Lotus XSL . . . . .	140
Figure 6-13. Lotus XSL Example . . . . .	141
Figure 6-14. Summary . . . . .	142
<b>Unit 7. VA Java Version 3 Persistence Builder Enhancements . . . . .</b>	<b>143</b>
Figure 7-2. Objectives. . . . .	144
Figure 7-3. Persistence Builder Review . . . . .	145
Figure 7-4. Key Elements. . . . .	146
Figure 7-5. Development Environment. . . . .	147
Figure 7-6. Runtime Environment . . . . .	148
Figure 7-7. Development Paths . . . . .	149
Figure 7-8. Multithreading (1). . . . .	150
Figure 7-9. Multithreading (2). . . . .	151
Figure 7-10. Connection Pooling . . . . .	152
Figure 7-11. Metamodel Validation. . . . .	153
Figure 7-12. Renaming Metamodel Elements . . . . .	154
Figure 7-13. Mapping for EJBs . . . . .	155
Figure 7-14. SQL Datatype Framework . . . . .	156
Figure 7-15. Summary . . . . .	157
<b>Unit 8. VA Java Version 3 Enhanced EJB Support. . . . .</b>	<b>159</b>
Figure 8-2. Objectives. . . . .	160
Figure 8-3. EJB - Enterprise JavaBeans (1) . . . . .	161
Figure 8-4. EJB - Enterprise JavaBeans (2)l . . . . .	162
Figure 8-5. EJB Development Environment. . . . .	163
Figure 8-6. New Features . . . . .	164
Figure 8-7. Two Forms of Inheritance. . . . .	165
Figure 8-8. EJB Inheritance Support (1)l . . . . .	166
Figure 8-9. EJB Inheritance Support (2). . . . .	167
Figure 8-10. EJB Inheritance Support (3). . . . .	168
Figure 8-11. EJB Inheritance Support (3). . . . .	169
Figure 8-12. Association Support (1) . . . . .	170
Figure 8-13. Association Support (2) . . . . .	171
Figure 8-14. Association Support (3) . . . . .	172

Figure 8-15. Properties Pane . . . . .	173
Figure 8-16. Create CMP Field SmartGuide . . . . .	174
Figure 8-17. Enhancements in Mapping Approach . . . . .	175
Figure 8-18. Top-down Approach . . . . .	176
Figure 8-19. Bottom-up Approach . . . . .	177
Figure 8-20. Meet-in-the-Middle Approach . . . . .	178
Figure 8-21. Create Default EJB Group SmartGuide . . . . .	179
Figure 8-22. Finder Helper Enhancement . . . . .	180
Figure 8-23. Implement WHERE Custom Finder . . . . .	181
Figure 8-24. Implement Method Custom Finder (1) . . . . .	182
Figure 8-25. Implement Method Custom Finder (2) . . . . .	183
Figure 8-26. Access Bean . . . . .	184
Figure 8-27. Beanified Wrapper . . . . .	185
Figure 8-28. Copy Helper . . . . .	186
Figure 8-29. Rowset . . . . .	187
Figure 8-30. Generating Access Bean (1) . . . . .	188
Figure 8-31. Generating Access Bean (2) . . . . .	189
Figure 8-32. Generating Access Bean (3) . . . . .	190
Figure 8-33. Deleting Access Bean . . . . .	191
Figure 8-34. Converter . . . . .	192
Figure 8-35. Instantiating Access Bean (1) . . . . .	193
Figure 8-36. Instantiating Access Bean (2) . . . . .	194
Figure 8-37. Advantage of Access Bean . . . . .	195
Figure 8-38. Starting EJB Server (1) . . . . .	196
Figure 8-39. Starting EJB Server (2) . . . . .	197
Figure 8-40. Exporting EJB to JAR file . . . . .	198
Figure 8-41. Summary . . . . .	199

## **Unit 9. VA Java Version 3 Enhanced Compiler & Distributed Debugger . . . . . 201**

Figure 9-2. Objectives . . . . .	202
Figure 9-3. HPJ Overview . . . . .	203
Figure 9-4. HPJ Diagram . . . . .	204
Figure 9-5. Differences in HPJ: VA Java V2 to V3 . . . . .	205
Figure 9-6. Calling HPJ within VA Java . . . . .	206
Figure 9-7. Distributed Debugger Overview . . . . .	207
Figure 9-8. Debugging Schema . . . . .	208
Figure 9-9. Languages Supported by Debugger . . . . .	209
Figure 9-10. Installing the Distributed Debugger . . . . .	210
Figure 9-11. Setup the Debugging Environment (1) . . . . .	211
Figure 9-12. Setup the Debugging Environment (2) . . . . .	212
Figure 9-13. Starting the Debug Process . . . . .	213
Figure 9-14. Debugger Start Dialog . . . . .	214
Figure 9-15. Attaching to Running Processes (1) . . . . .	215
Figure 9-16. Attaching to Running Processes (2) . . . . .	216
Figure 9-17. Attaching to Running Processes (3) . . . . .	217
Figure 9-18. Debugging an Applet . . . . .	218
Figure 9-19. Debug on Demand . . . . .	219
Figure 9-20. Breakpoints . . . . .	220
Figure 9-21. Setting Line Breakpoints . . . . .	221
Figure 9-22. Setting Line Breakpoints (2) . . . . .	222
Figure 9-23. Setting Method Breakpoints . . . . .	223
Figure 9-24. Setting a Method Breakpoint (2) . . . . .	224
Figure 9-25. View/Modify/Enable/Disable Breakpoints (1) . . . . .	225
Figure 9-26. View/Modify/Enable/Disable Breakpoints(2) . . . . .	226
Figure 9-27. Program Execution Control . . . . .	227



Figure 9-28. Inspecting Data . . . . .	228
Figure 9-29. Object Level Trace (OLT) . . . . .	229
Figure 9-30. OLT Schema . . . . .	230
Figure 9-31. OLT Step-by-Step . . . . .	231
Figure 9-32. Sample OLT Trace . . . . .	232
Figure 9-33. Summary . . . . .	233
<b>Unit 10. VA Java Version 3 Domino Access Builder . . . . .</b>	<b>235</b>
Figure 10-2. Objectives. . . . .	236
Figure 10-3. Domino Access Builder Introduction . . . . .	237
Figure 10-4. Domino Access Builder Features . . . . .	238
Figure 10-5. Requirements . . . . .	239
Figure 10-6. Installation . . . . .	240
Figure 10-7. Workspace Configuration . . . . .	241
Figure 10-8. Application Connections . . . . .	242
Figure 10-9. Creating a Connection Visually . . . . .	243
Figure 10-10. Fetching Data (Simple) . . . . .	244
Figure 10-11. Fetching Data Visually (Complex) . . . . .	245
Figure 10-12. Generating Domino Beans . . . . .	246
Figure 10-13. Setup SmartGuide to Generate Beans . . . . .	247
Figure 10-14. Select Database . . . . .	248
Figure 10-15. Select the Forms and Fields . . . . .	249
Figure 10-16. Complete the Generation of Beans . . . . .	250
Figure 10-17. Generated Beans . . . . .	251
Figure 10-18. Using Generated Beans . . . . .	252
Figure 10-19. Using Generated Beans . . . . .	253
Figure 10-20. Careful with Local Sessions . . . . .	254
Figure 10-21. Summary . . . . .	255
<b>Unit 11. VA Java Version 3 DB2 Stored Procedure Builder . . . . .</b>	<b>257</b>
Figure 11-2. Objectives. . . . .	258
Figure 11-3. Stored Procedures in DB2 V 6.1 . . . . .	259
Figure 11-4. Installing Stored Procedures . . . . .	260
Figure 11-5. VA Java and Stored Procedure Builder . . . . .	261
Figure 11-6. Features of Stored Procedure Builder . . . . .	262
Figure 11-7. Requirements . . . . .	263
Figure 11-8. Start Stored Procedure Builder . . . . .	264
Figure 11-9. Stored Procedure Builder GUI . . . . .	265
Figure 11-10. Configure Stored Procedure Builder . . . . .	266
Figure 11-11. Create the Stored Procedure . . . . .	267
Figure 11-12. Code Generator Parameters . . . . .	268
Figure 11-13. Define the Query (one or more) . . . . .	269
Figure 11-14. Conditions and Generated SQL . . . . .	270
Figure 11-15. Define Parameters and Options . . . . .	271
Figure 11-16. Generated Code . . . . .	272
Figure 11-17. Modifying Existing Code . . . . .	273
Figure 11-18. Run a Stored Procedure . . . . .	274
Figure 11-19. Integration with VA Java . . . . .	275
Figure 11-20. Using Stored Procedures . . . . .	276
Figure 11-21. Summary . . . . .	277
<b>Unit 12. VA Java Version 3 SQLJ Support . . . . .</b>	<b>279</b>
Figure 12-2. Objectives. . . . .	280
Figure 12-3. What is SQLJ? . . . . .	281
Figure 12-4. SQLJ Specification . . . . .	282
Figure 12-5. Using SQLJ . . . . .	283

Figure 12-6. Using SQLJ (2) . . . . .	284
Figure 12-7. Databases Supporting SQLJ . . . . .	285
Figure 12-8. Translating the SQLJ File . . . . .	286
Figure 12-9. Customizing the Profile . . . . .	287
Figure 12-10. Advantages of SQLJ . . . . .	288
Figure 12-11. Calling Stored Procedures using JDBC . . . . .	289
Figure 12-12. Calling Stored Procedures Using SQLJ . . . . .	290
Figure 12-13. VisualAge for Java Support for SQLJ . . . . .	291
Figure 12-14. Setting up VA Java SQLJ Support . . . . .	292
Figure 12-15. Using VA Java to Create the SQLJ file . . . . .	293
Figure 12-16. Editing the SQLJ File . . . . .	294
Figure 12-17. Commands for the SQLJ Editor (1) . . . . .	295
Figure 12-18. Commands for the SQLJ Editor (2) . . . . .	296
Figure 12-19. Commands for the SQLJ Editor (3) . . . . .	297
Figure 12-20. Translating the SQLJ File . . . . .	298
Figure 12-21. SQLJ Translator Output (1) . . . . .	299
Figure 12-22. SQLJ Translator Output (2) . . . . .	300
Figure 12-23. SQLJ Translator Output (3) . . . . .	301
Figure 12-24. Debugging SQLJ Code . . . . .	302
Figure 12-25. Summary . . . . .	303
<b>Unit 13. VA Java Version 3 Tool Integration API . . . . .</b>	<b>305</b>
Figure 13-2. Objectives . . . . .	306
Figure 13-3. Tool API . . . . .	307
Figure 13-4. Remote Access to Tool API . . . . .	308
Figure 13-5. Starting Remote Access Server . . . . .	309
Figure 13-6. Tool Servlet . . . . .	310
Figure 13-7. Developing Tool Servlet . . . . .	311
Figure 13-8. Debugging Tool Servlet . . . . .	312
Figure 13-9. Assigning Aliases to Tool Servlets . . . . .	313
Figure 13-10. Samples . . . . .	314
Figure 13-11. Summary . . . . .	315
<b>Unit 14. VA Java Version 3 Enterprise Access Builders Transactions . . . . .</b>	<b>317</b>
Figure 14-2. Objectives . . . . .	318
Figure 14-3. Common Connector Framework (CCF) . . . . .	319
Figure 14-4. Java Record Framework . . . . .	320
Figure 14-5. Record Type and Record . . . . .	321
Figure 14-6. EAB Command . . . . .	322
Figure 14-7. EAB Navigator . . . . .	323
Figure 14-8. Mapper and Business Object . . . . .	324
Figure 14-9. EAB Enhancements . . . . .	325
Figure 14-10. Enhanced Ease of Use . . . . .	326
Figure 14-11. Record Type Importers . . . . .	327
Figure 14-12. Import COBOL to Record Type . . . . .	328
Figure 14-13. Import 3270 to Record Type (1) . . . . .	329
Figure 14-14. Import 3270 to Record Type (2) . . . . .	330
Figure 14-15. Create Record Type SmartGuide . . . . .	331
Figure 14-16. Enhancements of Java Record(1) . . . . .	332
Figure 14-17. Enhancements of Java Record(2) . . . . .	333
Figure 14-18. Enhancements of Java Record(3) . . . . .	334
Figure 14-19. Enhancements of COBOL Record . . . . .	335
Figure 14-20. New Record Type Editor . . . . .	336
Figure 14-21. Create Record from Record Type SmartGuide . . . . .	337
Figure 14-22. Create Command SmartGuide . . . . .	338
Figure 14-23. New Command Editor . . . . .	339

Figure 14-24. Limitation of New Command Editor .....	340
Figure 14-25. Changes in EAB Command/Navigator. ....	341
Figure 14-26. Create Mapper SmartGuide .....	342
Figure 14-27. Enhancements of Mapper Editor .....	343
Figure 14-28. Mapper: Array Support Enhancement. ....	344
Figure 14-29. Mapper: More Enhancements. ....	345
Figure 14-30. EAB Session Bean Tool. ....	346
Figure 14-31. Create EAB Session Bean SmartGuide (1) .....	347
Figure 14-32. Create EAB Session Bean SmartGuide (2) .....	348
Figure 14-33. EAB Session Bean Editor .....	349
Figure 14-34. Developing EAB Session Bean .....	350
Figure 14-35. Environment Properties of EAB Session Bean .....	351
Figure 14-36. EAB Entity Bean. ....	352
Figure 14-37. IMS TOC Connector .....	353
Figure 14-38. MQSeries Connector .....	354
Figure 14-39. Host-on-Demand (HOD) Connector .....	355
Figure 14-40. SAP R/3 Connector .....	356
Figure 14-41. Limitations .....	357
Figure 14-42. Summary .....	358
<b>Unit 15. VA Java Version 3 Enhanced CICS Support with JCICS .....</b>	<b>359</b>
Figure 15-2. Objectives. ....	360
Figure 15-3. CICS and Java Roadmap .....	361
Figure 15-4. Running CICS Java Programs .....	362
Figure 15-5. JCICS Classes .....	363
Figure 15-6. Requirements .....	364
Figure 15-7. Simple JCICS Program. ....	365
Figure 15-8. Simple JCICS Program Code .....	366
Figure 15-9. ET/390 Setup .....	367
Figure 15-10. JavaInstall File .....	368
Figure 15-11. VA Java Host Sessions (1) .....	369
Figure 15-12. VA Java Host Sessions (2) .....	370
Figure 15-13. VA Java Host Sessions (3) .....	371
Figure 15-14. ET/390 Logon Data .....	372
Figure 15-15. ET/390 Project Properties (1) .....	373
Figure 15-16. ET/390 Project Properties (2) .....	374
Figure 15-17. ET/390 Project Properties (3) .....	375
Figure 15-18. Package Properties .....	376
Figure 15-19. Export and Bind Flow .....	377
Figure 15-20. Export and Bind .....	378
Figure 15-21. Executing the Program .....	379
Figure 15-22. Debugging CICS Java Applications .....	380
Figure 15-23. Visual Composition (1) .....	381
Figure 15-24. Visual Composition (2) .....	382
Figure 15-25. CICS IIOP .....	383
Figure 15-26. More Information .....	384
Figure 15-27. Summary .....	385
<b>Unit 16. VA Java Version 3 Extras (Technical Preview) .....</b>	<b>387</b>
Figure 16-2. Technology Previews. ....	388
Figure 16-3. Jax Jikes Application Extractor .....	389
Figure 16-4. Objectives (Jax) .....	390
Figure 16-5. What is Jax? .....	391
Figure 16-6. Advantages of Using Jax .....	392
Figure 16-7. JAX in Detail .....	393
Figure 16-8. Using Jax .....	394

Figure 16-9. Summary (Jax) . . . . .	395
Figure 16-10. Jinsight . . . . .	396
Figure 16-11. Objectives (Jinsight) . . . . .	397
Figure 16-12. Jinsight Definition . . . . .	398
Figure 16-13. Jinsight Background . . . . .	399
Figure 16-14. Installation of Jinsight . . . . .	400
Figure 16-15. Creating a Program Trace . . . . .	401
Figure 16-16. Uninstall Jinsight . . . . .	402
Figure 16-17. Summary (Jinsight) . . . . .	403
Figure 16-18. XMI Toolkit and XMI Bridge . . . . .	404
Figure 16-19. XMI Toolkit and XMI Bridge Overview . . . . .	405
<b>Unit 17. VA Java Version 3 Java 2 Support . . . . .</b>	<b>407</b>
Figure 17-2. Objectives . . . . .	408
Figure 17-3. Overview . . . . .	409
Figure 17-4. Overview (2) . . . . .	410
Figure 17-5. VA Java Migration Steps . . . . .	411
Figure 17-6. Java 2 Support at GA . . . . .	412
Figure 17-7. Summary . . . . .	413

---

## **Part 2. Exercises . . . . .415**

<b>Exercise E1. Using the IDE Enhancements . . . . .</b>	<b>417</b>
Part I: Using the new SmartGuide for Visual Applications . . . . .	418
Part II. Search/Replace, Keyword Completion, and Macro Generation . . . . .	419
Part III. Use the Resource Management . . . . .	420
Part IV. Use the New Filter Options . . . . .	420
Part V. Use the New Debugger Functions . . . . .	420
<b>Exercise E2. Accessing Relational Databases with Data Access Beans . . . . .</b>	<b>423</b>
Part I: Create a Simple GUI for the application . . . . .	424
Part II. Select Data from a Database Table . . . . .	424
Part III. Select a Column or Row from the Result Set . . . . .	425
Part IV. Retrieve a Single Value from the Result Set . . . . .	425
Part V. Optional: Add an Update Action . . . . .	426
Part VI. Optional: Add a Stored Procedure Call . . . . .	427
<b>Exercise E3. Servlet Builder with JSP . . . . .</b>	<b>429</b>
Part I: Create a Visual Servlet with an Employee Table . . . . .	430
Part II: Add Logic to Fill the Employees Table . . . . .	431
Part III: Transfer to Another Servlet . . . . .	431
Part IV: Transfer to a JSP . . . . .	433
<b>Exercise E4. XML Parser for Java . . . . .</b>	<b>437</b>
Part I: Create Document Type Definition and XML File . . . . .	438
Part II. Create an Applet to Invoke the XML Parser . . . . .	439
Part III: Lotus XSL Processor . . . . .	444
<b>Exercise E5. Developing and Using an EJB . . . . .</b>	<b>447</b>
Part I: Create a Container-Managed Entity Bean . . . . .	448
Part II. Create an Access Bean . . . . .	450
Part III. Use the Access Bean in a GUI . . . . .	450
Part IV. Use the Access Bean in a JSP and Servlet . . . . .	451
Part V. Optional Extension . . . . .	452

<b>Exercise E6. Using the HPJ Compiler and the Distributed Debugger. . . .</b>	<b>453</b>
Part I: Compile a Java Program with HPJ Inside VA Java . . . . .	454
Part II. Compile a Java Program with HPJ Outside VA Java . . . . .	454
Part III. Debug a Compiled Java Program. . . . .	454
Part IV. Debug a Interpreted Java Program . . . . .	455
<b>Exercise E7. Notes Access Builder . . . . .</b>	<b>457</b>
Part I: Verify the Connection . . . . .	458
Part II: Journal Database . . . . .	459
Part III. Generate an Access Bean Using Domino Access Builder . . . . .	460
Part IV. Deployment . . . . .	462
<b>Exercise E8. DB2 Stored Procedure Builder . . . . .</b>	<b>463</b>
Part I: Create a Stored Procedure. . . . .	463
Part II. Using a Stored Procedure. . . . .	464
<b>Exercise E9. Database Access with SQLJ . . . . .</b>	<b>467</b>
Part I: An SQLJ Program . . . . .	468
Part II: Test Program . . . . .	468
Part III: Servlet . . . . .	469
Part IV. Profiler . . . . .	469
<b>Exercise E10. Remote Tool API and Tool Servlet . . . . .</b>	<b>471</b>
Part I: Tool Servlet. . . . .	472
<b>Exercise E11. Enterprise Access Builder for Transactions . . . . .</b>	<b>473</b>
Part I: Import and Edit a COBOL Record . . . . .	474
Part II. Creating a Command . . . . .	475
Part III: Optional: Create an EAB Session EJB . . . . .	475
<hr/> <b>Part 3. Workshop Environment . . . . .</b>	<hr/> <b>477</b>
<b>A. Setup and Installation . . . . .</b>	<b>479</b>
Hardware Requirements . . . . .	479
Software Requirements . . . . .	479
Exercise Software. . . . .	480
Exercise Database . . . . .	480
<b>B. Special Notices. . . . .</b>	<b>481</b>
<b>C. Related Publications . . . . .</b>	<b>483</b>
International Technical Support Organization Publications . . . . .	483
Redbooks on CD-ROMs . . . . .	484
Other Publications . . . . .	484
<b>How to Get ITSO Redbooks . . . . .</b>	<b>485</b>
<b>List of Abbreviations . . . . .</b>	<b>487</b>
<b>ITSO Redbook Evaluation . . . . .</b>	<b>489</b>



---

# Instructor Course Overview

The goal of this workshop is to provide detailed information about VisualAge for Java Version 3.

The output are:

- ❑ a set of Freelance files (one per unit) containing all the foils used by the instructor
- ❑ a single PDF file containing the foils, speaker and student notes, and exercise instructions for the students

---

# Course Description

---

## VisualAge for Java Version 3

**Duration: 3 days**

### Purpose

This course is designed to provide students the skills required to use VisualAge for Java Version 3.

### Audience

Architects, designers, and developers of object-oriented applications.

### Prerequisites

Students are expected to have extensive skills in the following areas:

- ☐ Object oriented concepts, design and programming
- ☐ Java language
- ☐ VisualAge for Java Version 2
- ☐ Knowledge of WebSphere



---

## Agenda

	Time	Topic	Lab Exercise
<b>Day 1</b>			
	15	Welcome	
	20	Introduction	
	40	IDE Enhancements	
	60		LAB: IDE
	45	Data Access Beans	
	90		LAB: Data Access Beans
	30	WebSphere Integration	
	20	Servlet Builder	
	45		LAB: Servlet Builder
<b>Day 2</b>			
	15	XML Parser	
	60		LAB: XML Parser
	20	Persistence Builder	
	40	Enhanced EJBs	
	120		LAB: EJBs
	30	HPJ & Distributed Debugger	
	30		LAB: HPJ/Debugger
	30	Notes Access Builder	
			LAB: Notes/Domino
<b>Day 3</b>			
	30	DB2 Stored Procedures	
	45		LAB: Stored Procedure Builder
	20	SQLJ Support	
	30		LAB: SQLJ
	15	Remote Tool API	
	20		LAB: Remote Tool API
	30	EAB for Transactions	
	45		LAB: EAB Records, Commands
	20	JCICS Support	
	15	VA Java Extras (Tech. Preview)	
	15	Java 2 Preview	
	5	Conclusion	



# **Part 1**

# **Lectures**

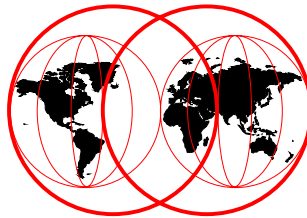


# 1 **VA Java Version 3**

## **Introduction and Overview**

**VisualAge for Java Version 3**

Introduction - Overview



# Objectives



## Prerequisites

## Agenda

- Presentations
- Experience the product with Lab Exercises

## VisualAge for Java Version 3

- Overview
- Features

## Issues

- DB2 Support

*Figure 1-2. Objectives*

## Prerequisites



### Attendees of this Workshop

- must have a good understanding of VisualAge for Java Version 2
- need Java language skills
- should be familiar with the JavaBeans concept
- want to receive a first hand detailed introduction to VA Java Version 3

**We created this workshop as a technical update class for users of VisualAge for Java Version 2.**

**VisualAge for Java Version 3 ships as a professional and an enterprise version.**

**Our goal is to provide you detailed information on the new features in the scope of the professional version and also cover the enhancements and new functions of the enterprise version.**

*Figure 1-3. Prerequisites*

# Agenda



Welcome

Introduction

IDE Enhancements

**LAB: IDE**

Data Access Beans

**LAB: Data Access Beans**

WebSphere Integration

Servlet Builder

**LAB: Servlet Builder**

XML Parser

**LAB: XML Parser**

Persistence Builder

Enhanced EJBs

**LAB: EJBs**

HPJ & Distr. Debugger

**LAB: HPJ/Debugger**

Domino Access Builder

**(LAB: Domino Access Builder)**

DB2 Stored Proc

**LAB: Stored Proc**

SQLJ Support

**LAB: SQLJ**

Remote Tool API

**LAB: Tool API**

EAB for Transactions

**LAB: EAB**

JCICS Support

VA Java Extras

Java 2

Figure 1-4. Agenda



## VA Java V3 Professional



	Win	OS/2	AIX	J2/NT	Linux
IDE/VCE	x	x	x	x	x
Tools API, Remote API	x	x	x	x	-rem
Documentation	x	x	x	x	x
Data Access Beans	x	x	x	x	
Stored Procedure Builder	NT				
SQLJ	x	x	x	x	
Remote Debugger (front/back)	x,NT			x	
JSP Tools	x				
WebSphere Test Environment	NT				
Entry Version (class limit)	x	x	x		x
JDK	117	117	116	12	117
Swing	103	103	103	11	103

Figure 1-5. VisualAge for Java Version 3 Professional

## VA Java V3 Enterprise



	NT	9x	OS/2	AIX	J2/NT
Remote Debugger (front/back)	X		X	X	X
JSP Tools	X			X	
WebSphere Test Environment	X			X	
EAB for Tx, Connectors	X	X	X	-hod	
JCICS	X				
EJB Tools	X			X	
IDL Development Env.	X	X	X	X	
ET/WS, 390, 400	X	X	WS	WS	
Data Access Builder	X	X	X		
Access Bld (RMI,C++,SAP,Dom)	X	X	-dom	-dom	
Servlet Builder	X	X	X	X	
Persistence Builder	X	X	X	X	
Team Support	X	X	X	X	X
Tech. Previews (jax,jinsight,xsl,...)	X	X		xsl/jax	

VA Java V3 - Introduction/Overview

© 1999 IBM Corporation

99SWB402UW

Figure 1-6. VisualAge for Java Version 3 Enterprise

## **VisualAge for Java Version 3 Enh.**



**IDE Enhancements**

**VCE Enhancements**

**WebSphere/Servlet Enhancements**

**Enterprise Enhancements**

**Debugger Enhancements**

**Integration Enhancements**

**Technology Previews**

*Figure 1-7. VisualAge for Java Version 3 Enhancements*

## IDE Enhancements



**Manipulate Project Resources from the IDE**

**Global Search/Replace**

**Customizable Key Bindings**

**Open on Selection**

**View Reference Help**

**Code Assist Enhancements (Ctrl + space)**

- keyword completions
- macros

*Figure 1-8. IDE Enhancements*

## Visual Composition Enhancements



**Create Application SmartGuide**

**Configure VCE code generation (inner classes)**

**Custom layout manager support**

- Improvements for GridBagLayout manager

**Reset button in Property sheet**

*Figure 1-9. Visual Composition Enhancements*

## WebSphere/Servlet Enhancements



### WebSphere Integration

- Servlet API 2.1 Support
- JSP Execution Monitor
- EJB Applications
- RMI over IIOP
- XML Applications

### Servlet Builder Enhancements

- JSP Call Wrapper Bean (call JSP/HTML)
- Variable Wrapper Bean (pass parms to JSP)
- Attribute Wrapper bean (servlet context)
- Servlet Wrapper (pass control to servlet)
- HTMLDiv

*Figure 1-10. WebSphere/Servlet Enhancements*

## Enterprise Enhancements (1)



### EJB Enhancements

- Entity EJB SmartGuide (also from existing tables, from EAB)
- Association/Inheritance EJB Mapping
- Access Beans (simplify programming)

### Enhanced Data Access

- DB2 Stored Procedures Builder
- SQLJ Development Environment
- Data Access Bean Enhancements
  - Procedure Bean, Modify Bean, WebSphere connection pooling
- VisualAge Persistence Enhancements
  - Multiple thread support, WebSphere connection pooling

*Figure 1-11. Enterprise Enhancements (1)*

## Enterprise Enhancements (2)



### EAB Enhancements

- New look-and-feel for EAB tooling
- New SmartGuide to generate records from 3270 streams

### Bundled e-Connectors

- CICS, MQSeries, HOD, Encina, SAP, and IMS ITOC

### CICS Enhancements

- CICS IIOP Development Support (two-tier CICS via IIOP)
- JCICS Development Support

### Domino Access Builder

- Generates beans for accessing databases, views, and forms
- Connects via IIOP (new in Domino V5) or Notes API

*Figure 1-12. Enterprise Enhancements (2)*



## Distributed Debugger Enhancements



### Separate Install

### New Java based UI

- In the V3 Professional today
- Enterprise Edition later this year  
(Synchronizing with 400 and 390 backends)

### Integration with IBM Object Level Trace

- Client/Server trace/performance analysis tool for WebSphere

### Explicit support for distributed debugging to Sun Solaris

*Figure 1-13. Distributed Debugger Enhancements*

## Integration Enhancements



### Remote Access to Tools API

- Allows external tools to invoke tools API
- WebSphere Studio V3 uses this to move files

### XMI Toolkit

- Easily work with Rose models in VA Java
- Compare differences between model and implementation from VA Java
- Synchronize model with implementation

*Figure 1-14. Integration Enhancements*

## Technology Previews



### **JAX (Java Application Extractor)**

- Reduces application size

### **Jinsight**

- Java profiler/performance analyzer

### **XMI Toolkit and XMI Bridge**

- Bridge between Rational Rose and VA Java/Persistence Builder/EJB

### **Lotus XSL Processor**

- XML to HTML converter

### **WebDAV Client (WWW Distributed Authoring and Versioning)**

### **JavaPureCheck (from Sun)**

### **Rioja (Record I/O for Java Applications)**

- API extends the Record Framework of VA Java

### **XML Productivity Kit**

- Java Beans from XML data files, Wiring JavaBeans that process XML

*Figure 1-15. Technology Previews*

## DB2 Issues



**Be sure to check what level of DB2 is supported**

### **Current driver:**

- Requires DB2 5.2 Fixpack 8
- DB2 5.2 Fixpack 10 requires additional fixes
- DB2 5.2 Fixpack 11 now available
- DB2 6.1 Fixpack 1 requires additional fixes
- DB2 6.1 Fixpack 1a now available

### **DB2 Stored Procedures Builder**

- Requires DB2 6.1

*Figure 1-16. DB2 Issues*

## Summary



### **VisualAge for Java is the prime Java Development Environment**

- V3 adds many new "productivity enhancements" for the Java developer
- IDE/VCE/Debugger enhancements

### **VisualAge for Java still has the best "Visual Builder in the Planet"**

**No one can beat the Enterprise Connectivity  
VisualAge Version 3.0 provides!**

*Figure 1-17. Summary*

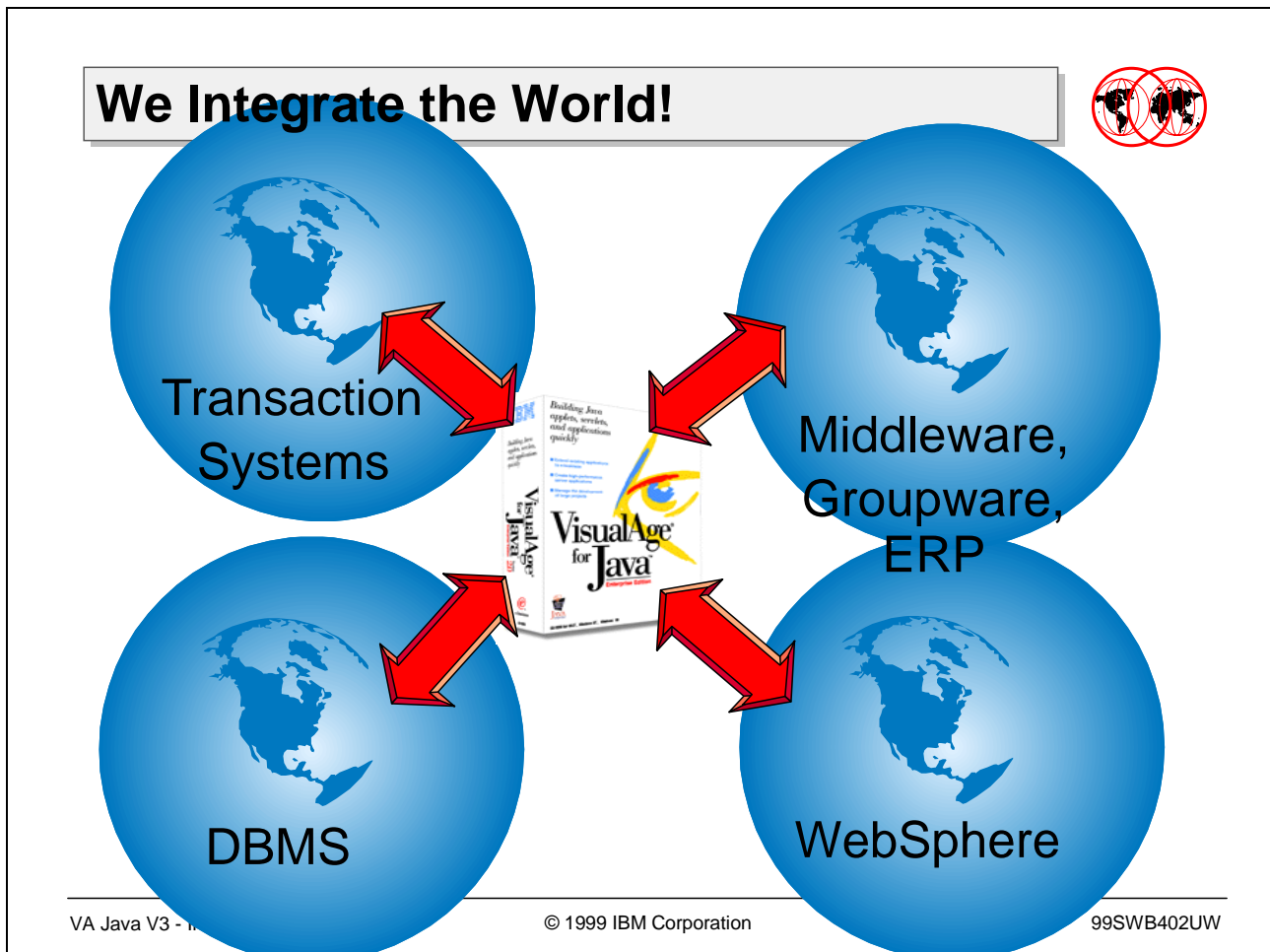
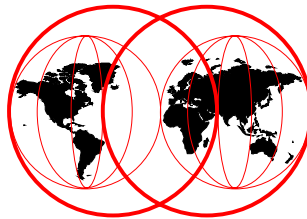


Figure 1-18. We Integrate the World!

## **2 VA Java Version 3 IDE Enhancements**

**VisualAge for Java Version 3**

IDE Enhancements



© 1999 IBM Corporation

99SWB402UW

## Objectives



### Understand the IDE Enhancements in VisualAge for Java V3

- **Enhanced ease of use**
  - Reference Help, Code Assist Feature
  - Search/Replace, Filter
  - Project Resources
  - Customizable Key Bindings
- **Code Generation**
  - Required Methods, Accessor Methods
- **New SmartGuide for Visual Application**
- **Visual Composition Editor (VCE) Enhancements**
- **BeanInfo Enhancements**
- **Debugger Enhancements**

*Figure 2-2. Objectives*

The IDE of VA Java Version 3 has numerous small enhancements that make it easier to develop applications.



## Help: View Reference Help



### Open the reference document for a selected program element

- Program element : Java keyword, Package, Class, Interface, Method
- Context Sensitive
- Select **View Reference Help** from popup menu

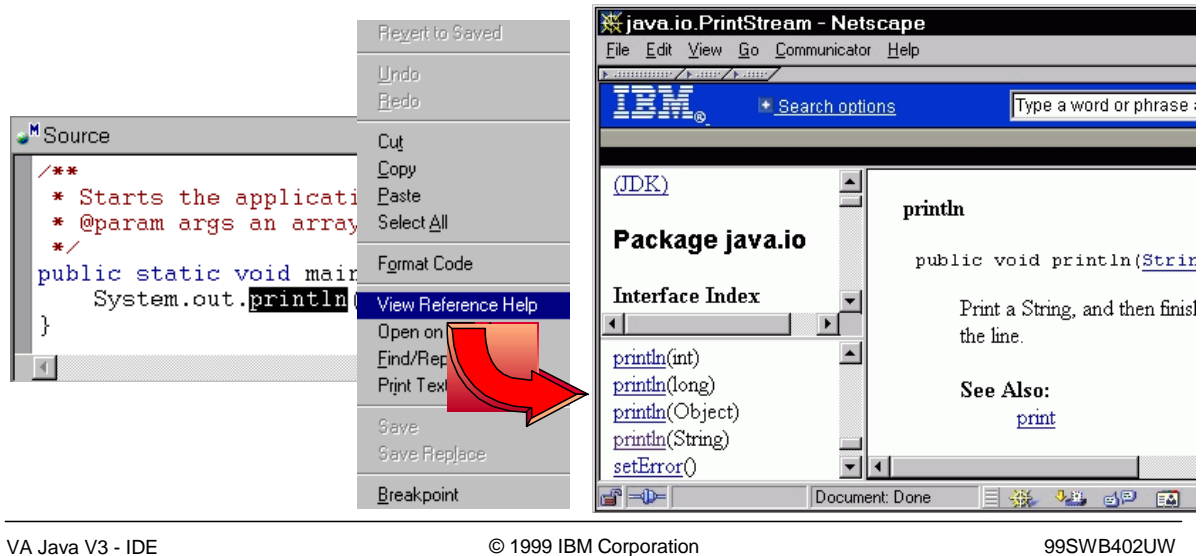


Figure 2-3. Help: View Reference Help

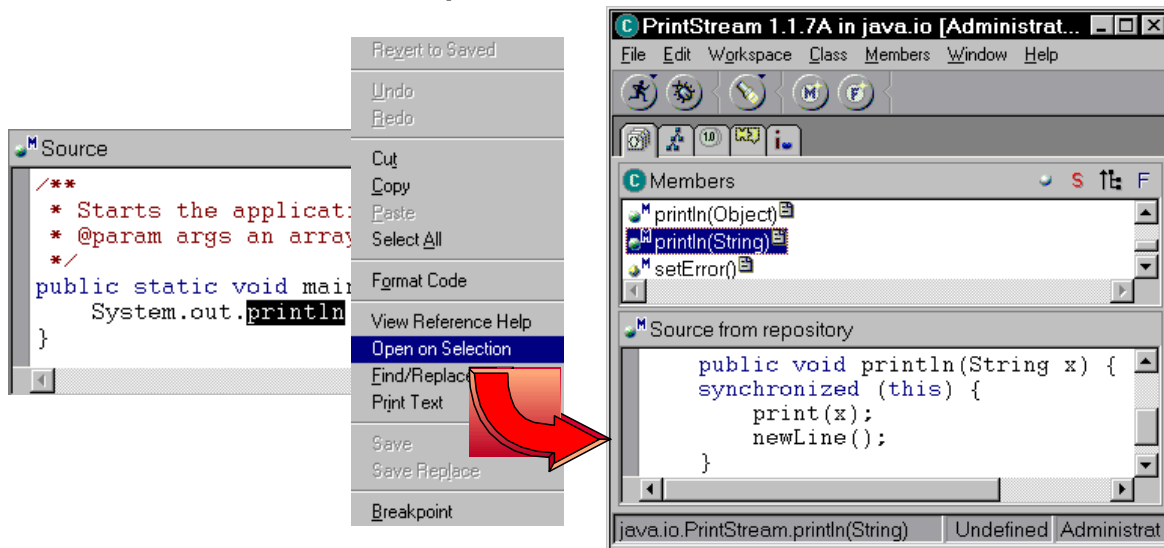
The View Reference Help is a fast-track jump into the VA Java Reference Help facility.

## Help: Open on Selection



### Open the browser window for a selected program element

- Program element : Package, Class, Interface, Method
- Context Sensitive
- Push **F3** Key or Select **Open on Selection** from popup menu



VA Java V3 - IDE

© 1999 IBM Corporation

99SWB402UW

Figure 2-4. Help: Open on Selection

With this facility you select a class or method (or ...) and open the VisualAge browser on that program element. This is a fast way to check out the features of a bean/class that you are using.

## Extended Code Assist Feature



### **Code Assist Feature (Ctrl+Space Key) is enhanced:**

- Class, Method, Field (same as VA Java V2)
- Keyword Completions
  - if - else, for, while, try - catch, ....
- Macros
  - JavaDoc comments for class/method, and tags for <user>/<timestamp>
- Note:
  - If previous source lines contain errors, Code Assist Feature will not work

### **Adding/Editing/Removing Keyword Completions and Macros**

- Use **Options Dialog : Coding -> Keyword Completions / Macros**
  - In Completion, "<|>" indicate the position of cursor

### **You can customize more ...**

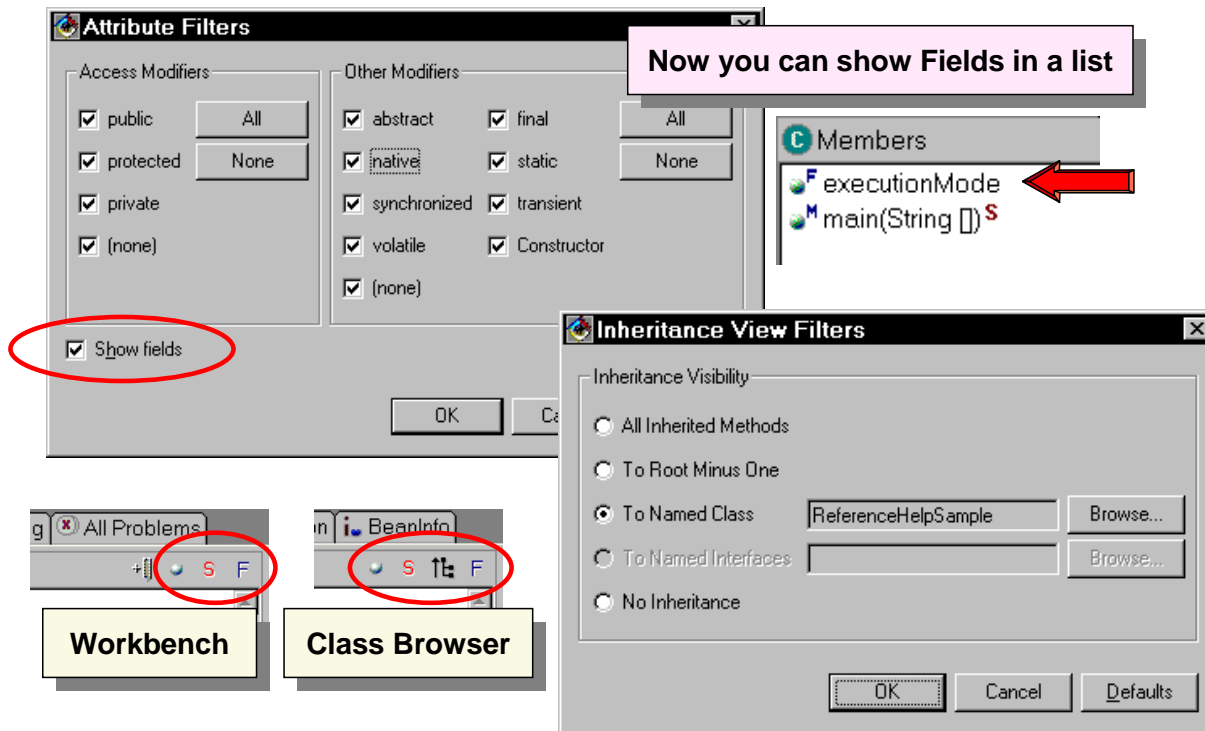
- Use **Options Dialog: Coding -> Code Assist**

*Figure 2-5. Extended Code Assist Feature*

The code assist feature (of VA Java Version 2) has a few more tricks.

You can define your own keyword completion phrases and macros.

## Enhanced Filters



VA Java V3 - IDE

© 1999 IBM Corporation

99SWB402UW

Figure 2-6. Enhanced Filters

The new filters allow you to tailor what program elements you see in the browser. Use *Members -> Attribute Filters* or *Members -> Inheritance Filters* in the project window, or *Selected -> Attributes Filters* in the Workbench.

For example, you can now see the fields as separate items, and select what kind of fields you want to see.

The browser provides icons (top-right) where you can easily switch between seeing fields or not, static and/or public members, and inherited methods.

## Generating Required Methods

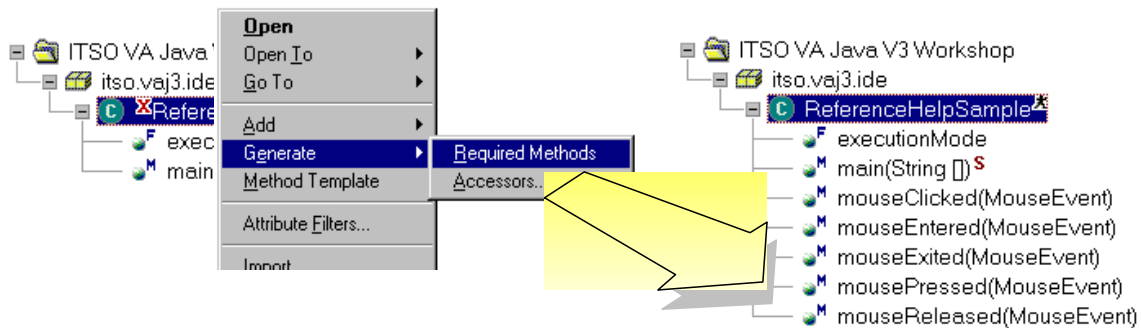


### Required Methods:

- Methods defined as abstract in the super class
- Methods needed to implement interfaces

### To generate stub methods:

- Select the class in **Workbench** and select **Generate -> Required Methods** from popup menu



VA Java V3 - IDE

© 1999 IBM Corporation

99SWB402UW

Figure 2-7. Generating Required Methods

When you create a new class you can have stub methods generated for methods that you must implement.

You can also add the stubs later when adding new interfaces to the class. This makes sure that you have the signature correct for your implementations.

## Generating Accessors

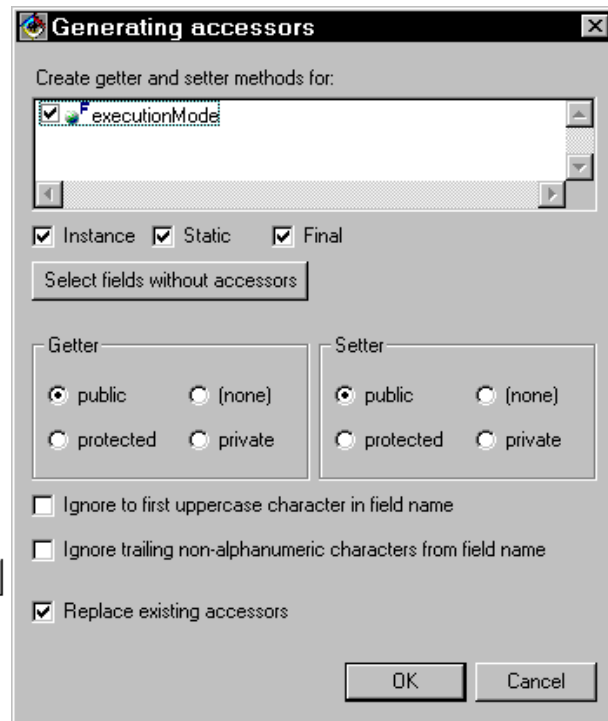
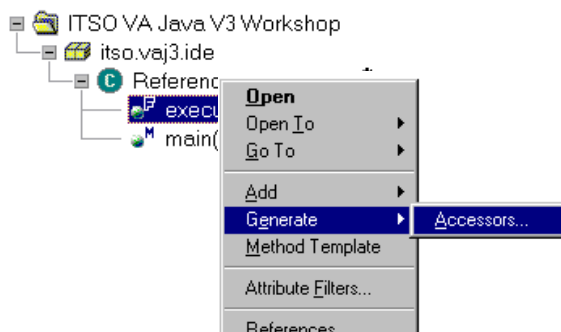


### Accessor Methods:

- getXXX() / setXXX()

### To generate accessors:

- Select the field and select **Generate -> Accessors** from popup menu



VA Java V3 - IDE

© 1999 IBM Corporation

99SWB402UW

Figure 2-8. Generating Accessors

For fields you can have accessor methods generated at any time.

## Enhanced Search/Replace



### Global Find/Replace

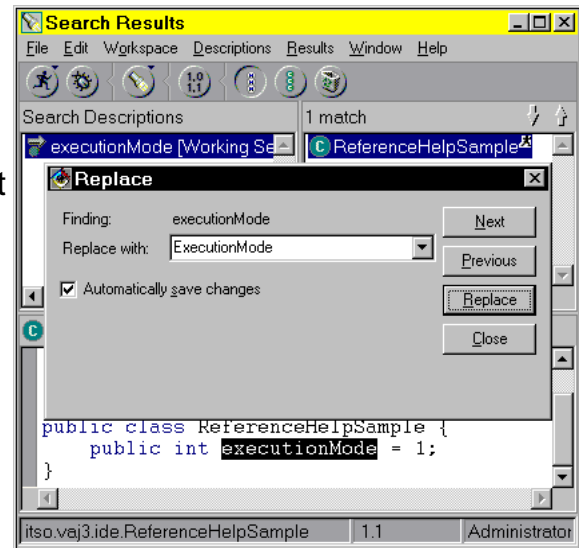
- Searching for text through the Workspace/Working Set and Replacing it
- Select **Workspace -> Text Search/Replace...** from the menu

### Global Find/Replace consist of:

- Searching for text
- Replacing text in the Search Result

### You can Replace text in any Search Results

- Select **Results -> Replace Matches...** from the menu of **Search Results** Window



VA Java V3 - IDE

© 1999 IBM Corporation

99SWB402UW

Figure 2-9. Enhanced Search/Replace

The global find/replace features make it easy to update references to program elements that have changed. For example you renamed a package and have to update all classes that reference the changed package.

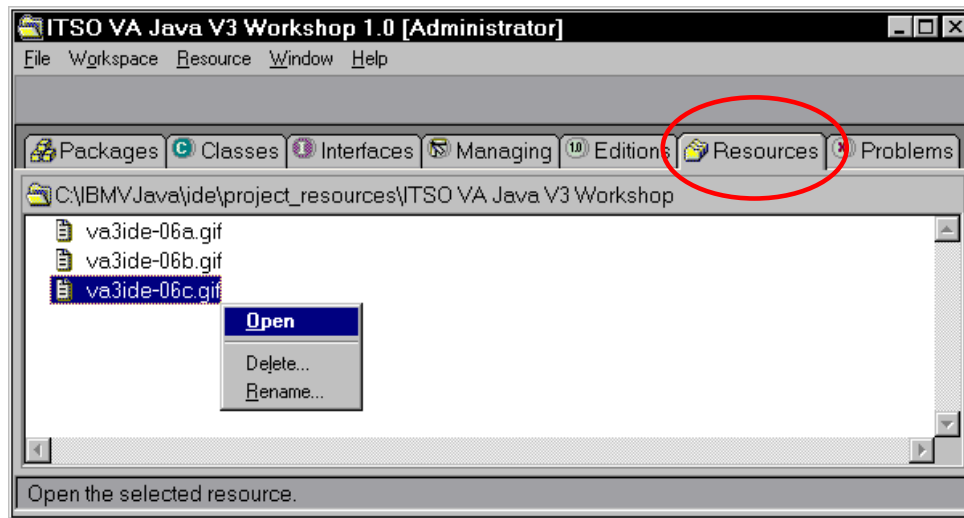
An example for the future: changing the referenced to Swing classes from `com.sun.xxx` to `javax.xxx`.

## Maintain Project Resources in VA Java



### Resources page is added to Project Browser

- You can edit/delete/rename resources



- To define special associations, use Window -> Options Dialog  
– **Resources -> Resource Associations**

Figure 2-10. Maintain Project Resources in VA Java

The project window has a new pane to maintain resources. On this pane you can see all the files stored in the IBMVJava\ide\project\_resources\...project... directory.

The window only supports simple operations, but not drag/drop from other file management windows.

You can define associations for file types and specify what program should be invoked when double-clicking on a file. Use *Window -> Options -> Resource -> Resource Associations*.



# New SmartGuide for Visual Application



## Create Application SmartGuide provides the easy way to create a visual application

- Swing based or AWT based application with:
  - Menu Bar (customizable)
  - Tool Bar (customizable)
  - Status Bar
  - Splash Screen
  - About Dialog
- You can start developing from generated code

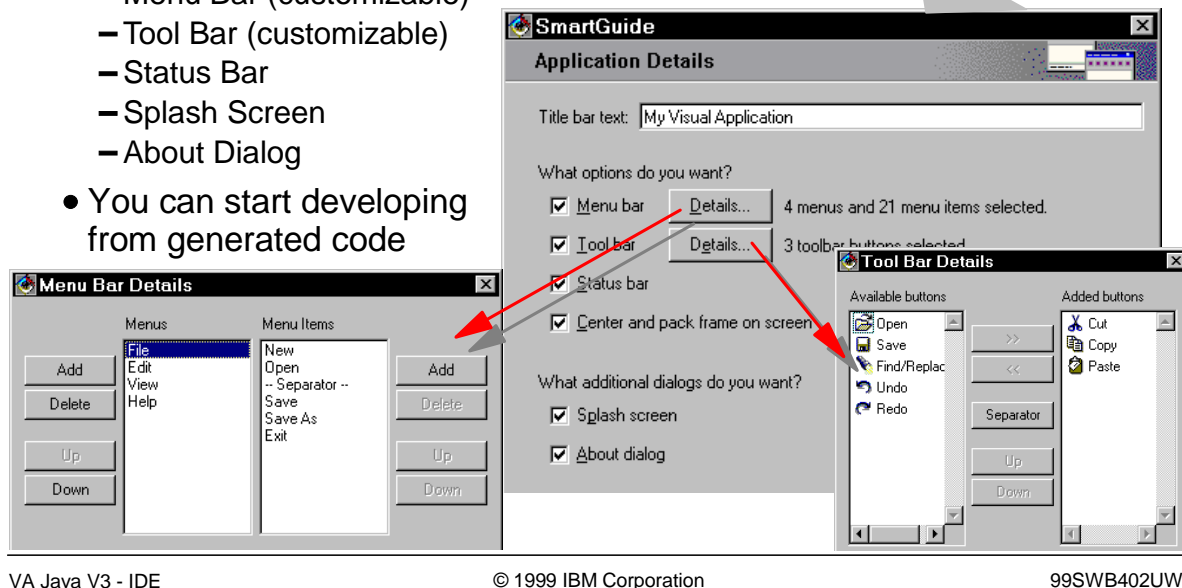
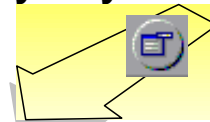


Figure 2-11. New SmartGuide for Visual Application

A new SmartGuide allows the creation of a complete GUI application. The SmartGuide can be invoked through an icon on the tool bar.

You can specify the menu bar, tool bar, status bar that you want attached to the application.

You can have a Splash dialog (a picture) displayed before the application starts, and you can have an About dialog accessible through the Help menu. These dialogs are separate classes that are generated.

## VCE: Layout Management



### Customized Layout Manager Support

- You can use a custom Layout Managers in Visual Composition
- Custom Layout Managers are automatically shown in the list of layout property
- Custom Layout Manager samples:
  - Project: IBM Java Examples
  - Package: com.ibm.ivj.examples.vc.customLayoutManager

### GridBag Layout Improvement

- Position/Size of components are kept, even if you change the layout to GridBag
- The easiest way using GridBag is:
  - Layout components using <null> Layout, and change the layout to GridBag
- **Layout Options** menu:
  - This menu is shown in popup menu of components
  - Use this menu to customize constraints of the component

*Figure 2-12. VCE: Layout Management*

Layout Manager improvements include support for custom layout managers.

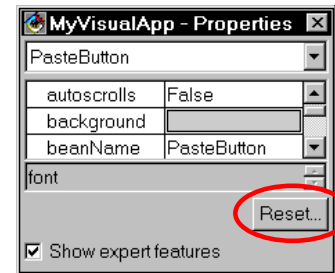
A big improvements is the better support for the GridBagLayout. You can now design a GUI without a layout manager, and then switch to GridBag and the layout is unchanged. Then you can use the GridBag constraints to make the layout more flexible and nicer.

## VCE: Property Settings and Editor



### Reset property settings

- To reset settings, click the **Reset...** button in the property window



### Using a custom property editor globally

- Create `ivj-property-editor-registry.properties` file in `\IBMVJava\IDE\program\lib` directory
- To register a custom editor:
  - `com.sun.java.swing.Border=myorg.mypkg.MyBorderEditor`
- To unregister a editor:
  - `com.sun.java.swing.Border=`
- You cannot change the editor for `java.lang.String`
- Already opened VCE will not be affected, event if you edit the file
  - When each VCE is opened, this file is checked

*Figure 2-13. VCE: Property Settings and Editor*

Improvements for property settings include a Reset button and better facilities for custom property managers.

## VCE: Swing Improvements



### Improved Swing Migration

- You can change a super class
  - from Frame/Applet/Panel/Dialog
  - to JFrame/JApplet/JPanel/JDialog
- VCE automatically migrates VC information
- But, you have to:
  - Change the super class of contained components, such as Button/List/...
  - Re-generate code

### Look & Feel in VCE

- You can change the default Look & Feel in VCE
- Create `swing.properties` file in `\IBMJava\IDE\program\lib` directory, and specify Look & Feel
  - `swing.defaultlaf=com.sun.java.swing.plaf.windows.WindowsLookAndFeel`
- Already opened VCE will not be affected, event if you edit the file
  - When each VCE is opened, this file is checked

*Figure 2-14. VCE: Swing Improvements*

It is now easier to migrate from AWT to Swing. You can change a super class and VCE automatically migrates the code to match. You have to manually change the superclasses of the contained components.

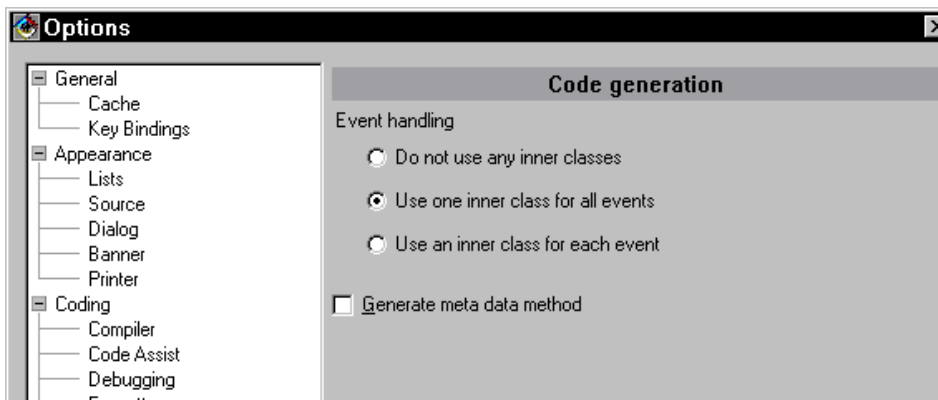
The Swing look-and-feel can now also be set for the VCE through a properties file.

## VCE: Code Generation Options



**You can specify how Visual Composition generates code for event handling**

- No inner classes (same style with VA Java V1&V2)
- One inner class for all events (Default)
- An inner class for each event



- Options Dialog : **Visual Composition -> Code Generation**

*Figure 2-15. VCE: Code Generation Options*

The code that is generated from the VCE for a class can now be influenced. In Version 2 no inner classes were used. In Version 3 you can choose if you want code generated with or without inner classes.

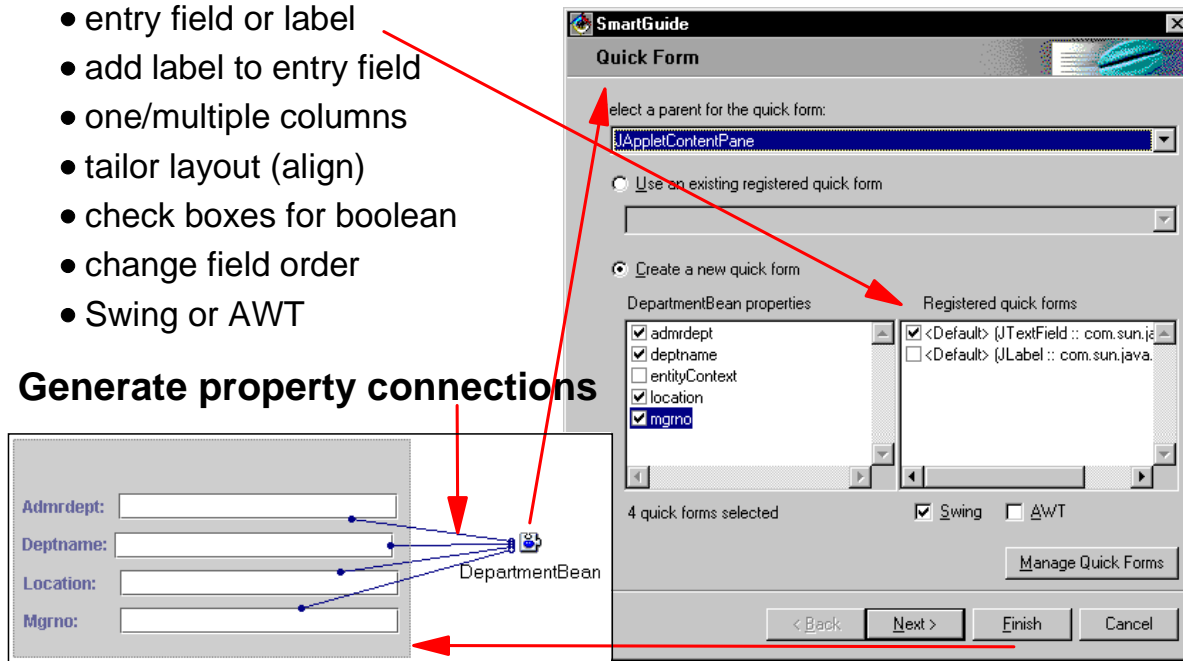
## VCE: Quick Form



### Quickly layout a GUI panel from properties of a JavaBean

- entry field or label
- add label to entry field
- one/multiple columns
- tailor layout (align)
- check boxes for boolean
- change field order
- Swing or AWT

### Generate property connections



VA Java V3 - IDE

© 1999 IBM Corporation

99SWB402UW

Figure 2-16. VCE: Quick Form

A new SmartGuide enables you to layout a GUI panel from an existing JavaBean, including the property-property connections between the panel components and the bean.

For each property you can decide to:

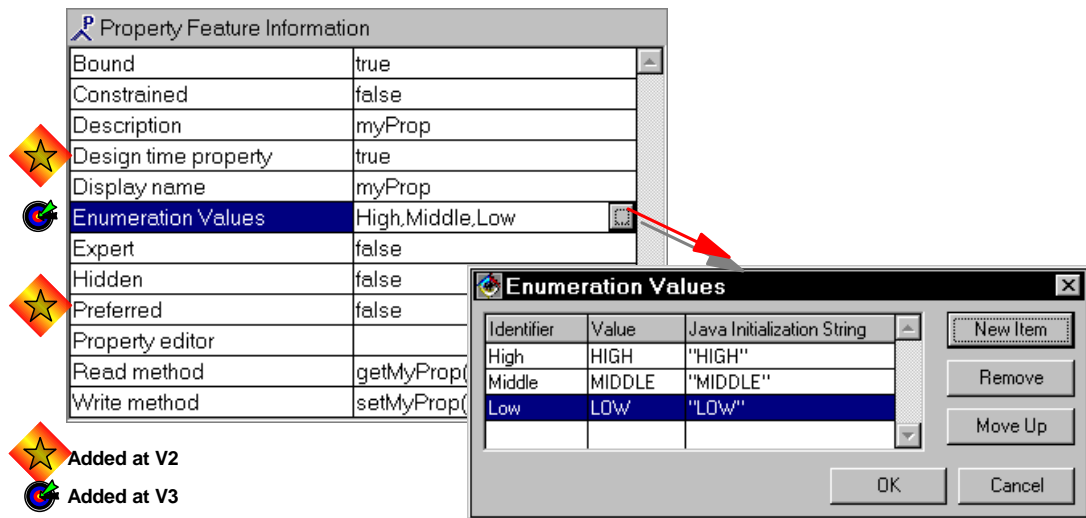
- ☐ include it or not
- ☐ entry field or label or check box (for booleans)
- ☐ generate a label for entry fields
- ☐ align into one or multiple columns
- ☐ change the order of the fields

## BeanInfo: Enumeration Values Prompt



### Enumeration Values Prompter

- You can define enumeration values for a property
- Enumeration values are stored in BeanInfo class, and used in VCE to select a value of the property



VA Java V3 - IDE

© 1999 IBM Corporation

99SWB402UW

Figure 2-17. BeanInfo: Enumeration Values Prompt

Enumeration values can now be specified for the attribute of a bean.

## BeanInfo: Delete Features



### In V2, you have to manually delete:

- a field which is used to store the property value
- a field which is used to maintain event listeners
- fireXXX() methods which is used to notify event

### In V3, you can delete them with features

- No need to delete them manually

### But, you still have to delete:

- fields/methods which are used to maintain bound/constrained property listeners
- methods which are used to notify propertyChange/vetoableChange event

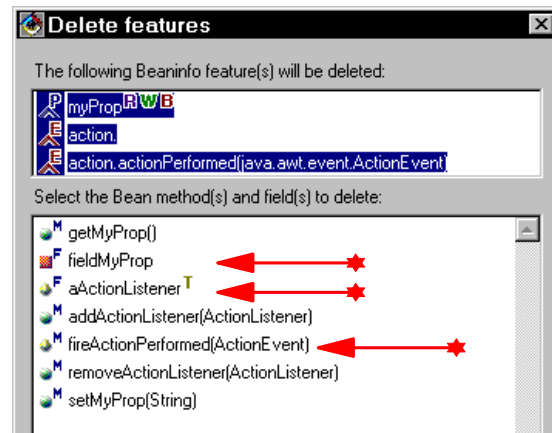


Figure 2-18. BeanInfo: Delete Features

In Version 2 when you deleted a property you could have the accessor methods deleted as well, but the field would stick around and to be deleted manually.

In Version 3 you can have an automatic complete cleanup.

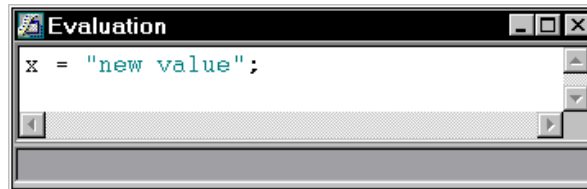


## Debugger Enhancements (1)



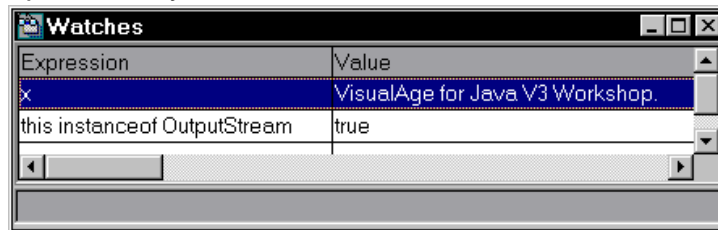
### Evaluation Area Window

- Use this window to type in evaluation codes and execute them
- Evaluation codes are kept, even if you terminate VA Java
- No need to type in evaluation code in the Code Pane every time



### Watches Window

- Use this window to watch values of variables/expressions
- Settings are kept, even if you terminate VA Java



VA Java V3 - IDE

© 1999 IBM Corporation

99SWB402UW

Figure 2-19. Debugger Enhancements (1)

The debugger provides two new windows.

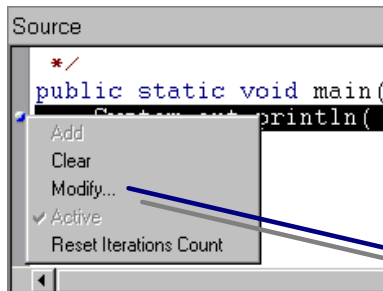
The Evaluation window can be used to enter Java code and execute it without disturbing the method you are debugging.

You can add selected variables to the Watches window and see how the values change during execution of the program. The variables in the list are saved when VA Java is terminated.

## Debugger Enhancements (2)

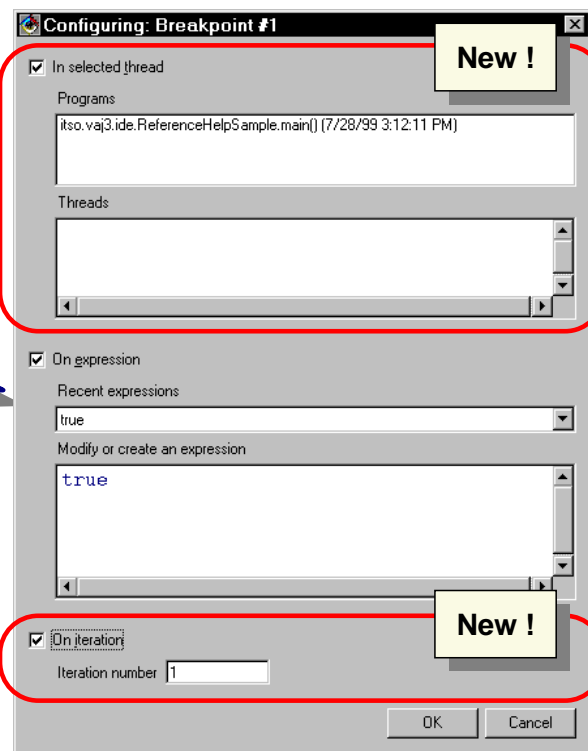


### Extended Breakpoint Settings



#### You can specify:

- Thread
- Condition (Expression)
- Iteration Count
  - You can reset iteration count from popup menu



VA Java V3 - IDE

© 1999 IBM Corporation

99SWB402UW

Figure 2-20. Debugger Enhancements (2)

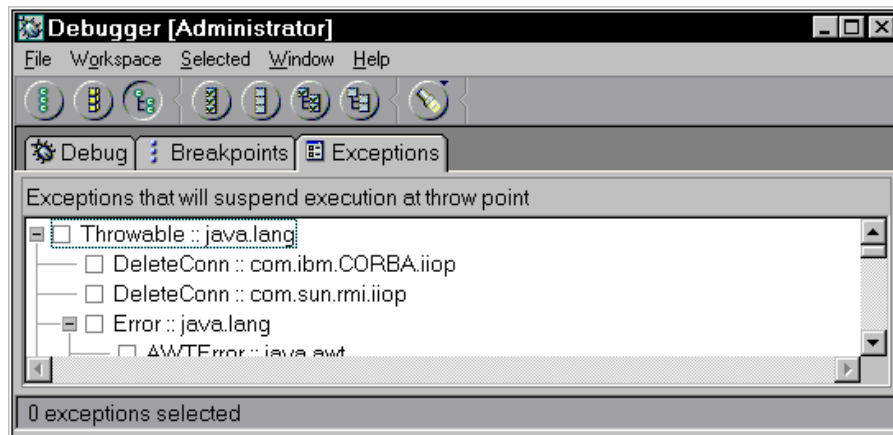
The specification of breakpoints is enhanced as well. For example, a breakpoint can be activated at a specific iteration count in a loop.

## Debugger Enhancements (3)



### Exceptions Page

- Enhancement of **Caught Exceptions** Dialog in VA Java V2
- You can use Tool Bar to sort/select exceptions



### External .class File Breakpoints

- In V3, external .class file breakpoints are shown in **Breakpoints Page**

Figure 2-21. Debugger Enhancements (3)

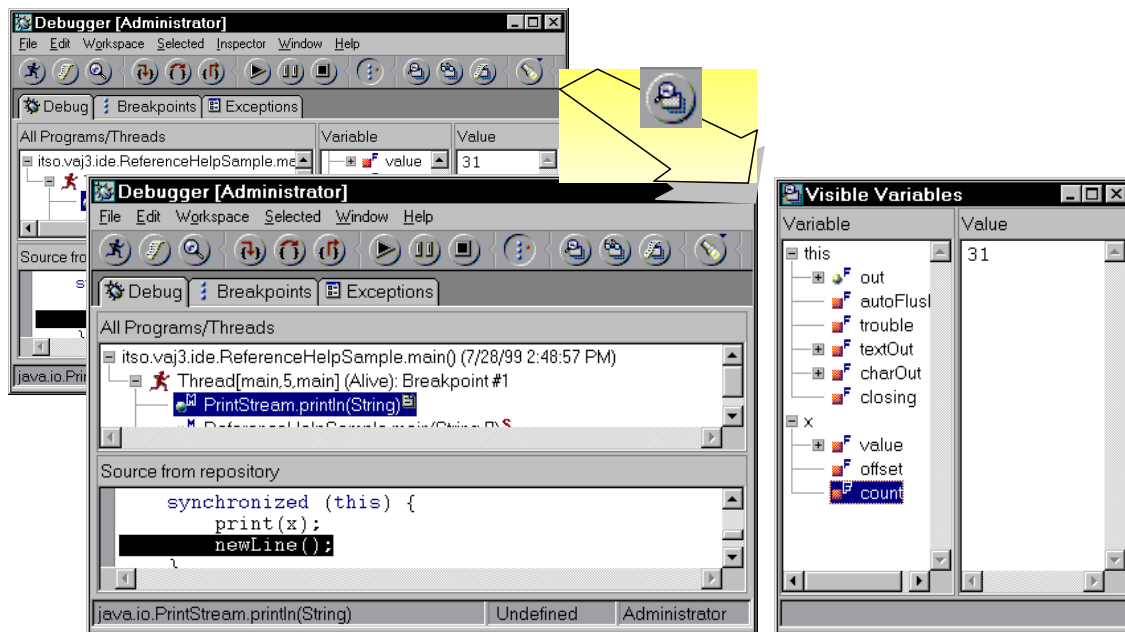
This debugger enhancement deals with exceptions. It is now easier to sort out the exceptions that occurred in a program.

## Debugger Enhancements (4)



### Visible Variables Window

- You can view visible variables in the separated window



VA Java V3 - IDE

© 1999 IBM Corporation

99SWB402UW

Figure 2-22. Debugger Enhancements (4)

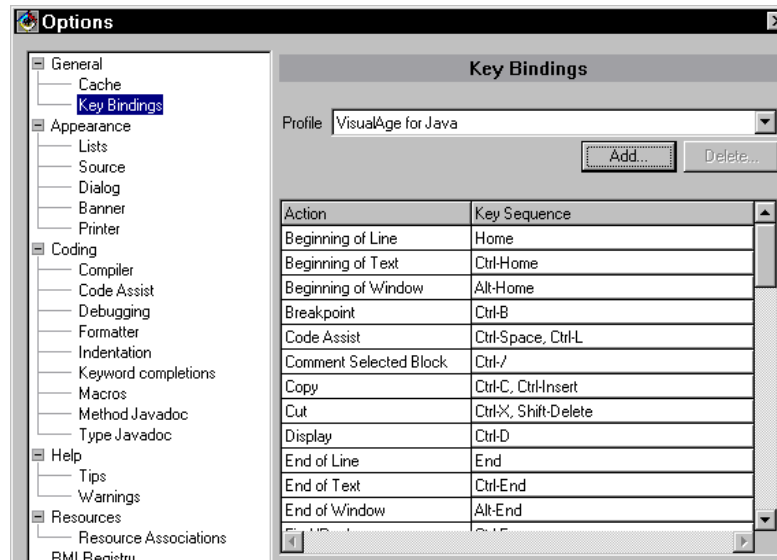
The Visible Variables window gives you easier access to all the visible variables and their values. The standard debugger window panes are very small for that purpose. The Visible Variables window is optional, when closed it goes back to the standard panes in the debugger window.

# Customizable Key Bindings



## Key Bindings become customizable

- Use Options Dialog: General -> Key Bindings
- Provided profiles: VisualAge for Java, Emacs



VA Java V3 - IDE

© 1999 IBM Corporation

99SWB402UW

Figure 2-23. Customizable Key Bindings

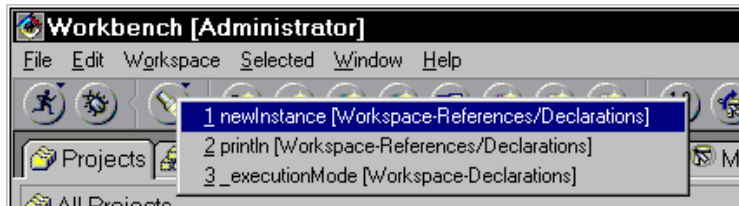
This new dialog allows you to change the actions for key strokes.

## History, Web-Browser like Interface



### Run/Search History

- You can retrieve them by clicking the button with Mouse Button 2
- ▼ mark above right of buttons indicate that the history function is available



### Web-Browser like user-interface

- Program elements are browsed in one window
- To activate this user-interface, use Options Dialog: *General*

Use buttons to go to  
Forward/Backward

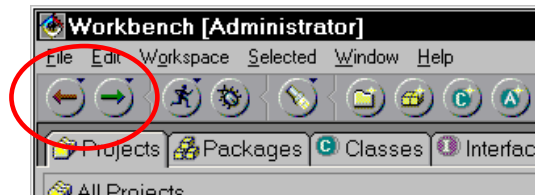


Figure 2-24. History, Web-Browser like Interface

The history of searches is kept and can easily be retrieved.

Instead of using multiple windows to browse program elements you can work with one window and use the back/forward buttons to switch between program elements.

## Summary



### **VisualAge for Java IDE becomes:**

- easier to use and more helpful
- more productive
- more customizable
- more powerful

*Figure 2-25. Summary*

**The VisualAge for Java IDE - better in every Version!**

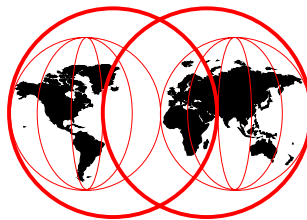




# **3 VA Java Version 3 Enhanced Data Access Beans**

**VisualAge for Java Version 3**

Data Access Beans



---

© 1999 IBM Corporation

## Objectives



### Enhanced Data Access Beans

### Review of the Select and Navigator beans

### Overview of the 6 new beans

- Modify bean
- Procedure call bean
- Selector beans
  - CellSelector
  - CellRangeSelector
  - RowSelector
  - ColumnSelector

### How to use the new beans

VA Java V3 - Data Access Beans

© 1999 IBM Corporation

99SWB402UW

*Figure 3-2. Objectives*

In this unit we have a look at the enhancements of the Data Access Beans.

VA Java Version 2 provided two beans: Select and Navigator

VA Java Version 3 provides 6 new bean for enhanced access to relational databases:

- ☐ Modify bean for tailored updates (INSERT, DELETE, UPDATE statements)
- ☐ Procedure Call bean to invoke a stored procedure
- ☐ 4 beans for access to specific rows, columns, and cells within a result set of a Select bean

## Data Access Beans Feature



### To use the Data Access Beans:

- Import the db2java.zip file located in \SQLLIB\JAVA to VA Java or  
add the db2java.zip to the workspace class path  
– Window -> Options -> Resources
- add the Data Access Beans feature to the workspace  
– Data Access Beans 3.0 (F2 - QuickStart menu)
- The beans are now accessible in the beans palette in the Database category

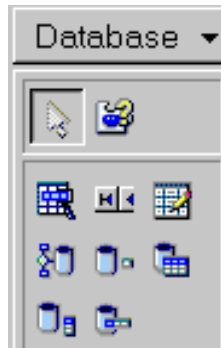


Figure 3-3. Data Access Beans Feature

The Data Access Beans feature must have access to the JDBC classes of the database system.

Either import the classes into the Workbench (make a new project) or add the zip/jar file to the class path specification. Use the Window -> Options menu and look for the Resources.

## Select Bean



**The Select bean provides a set of methods for relational database access:**

- set parameters (host variables)
- execute
- get column values
- next row
- new row, update row, delete row
- commit, rollback

**To access relational data using a Select bean, you connect user interface components to the Select bean**

**No major changes are made from VA Java 2.0**

**New option to remove schema prefix from SQL statements**

- select [userid.]employee.empno, ... from [userid.]employee  
– better portability

**New option to use WebSphere connection pools**

*Figure 3-4. Select Bean*

The Select bean is still the same as in VA Java Version 2.

## Navigator Bean



**The DBNavigator bean is a visual bean that you use with a Select bean to access data in a relational database**

**The DBNavigator bean provides a set of buttons that invoke the actions (methods) of the Select bean**

- Swing component

**No changes are made from VA Java 2.0**

*Figure 3-5. Navigator Bean*

The Navigator bean is still the same as in VA Java Version 2.

## Modify Bean



**Use the Modify bean for inserting, updating or deleting data in a database *WITHOUT* doing a query**

**The properties of the Modify bean are:**

- action property (connect to database and SQL statement)
- properties to get information such as whether the SQL statement has been executed and how many rows in the database were affected by it

**The methods provided by the Modify bean are:**

- execute, commit, rollback

**The Modify bean can execute every valid SQL statement, but it does *NOT* give access to a result set**

- you cannot use the Navigator bean together with the Modify bean!
  - use an event (button) to execute the Modify bean
- For access to a result set, you should use the Select bean or the Procedure Call bean

*Figure 3-6. Modify Bean*

In VA Java Version 2, all updates had to be done within a result set of a Select bean.

Version 3 provides the Modify bean to build SQL statements for updating the database. You can build INSERT, DELETE, and UPDATE statements.

## Editing the Modify Bean Properties



The action property lets you to define a DB access class with

- SQL statements
- Connections

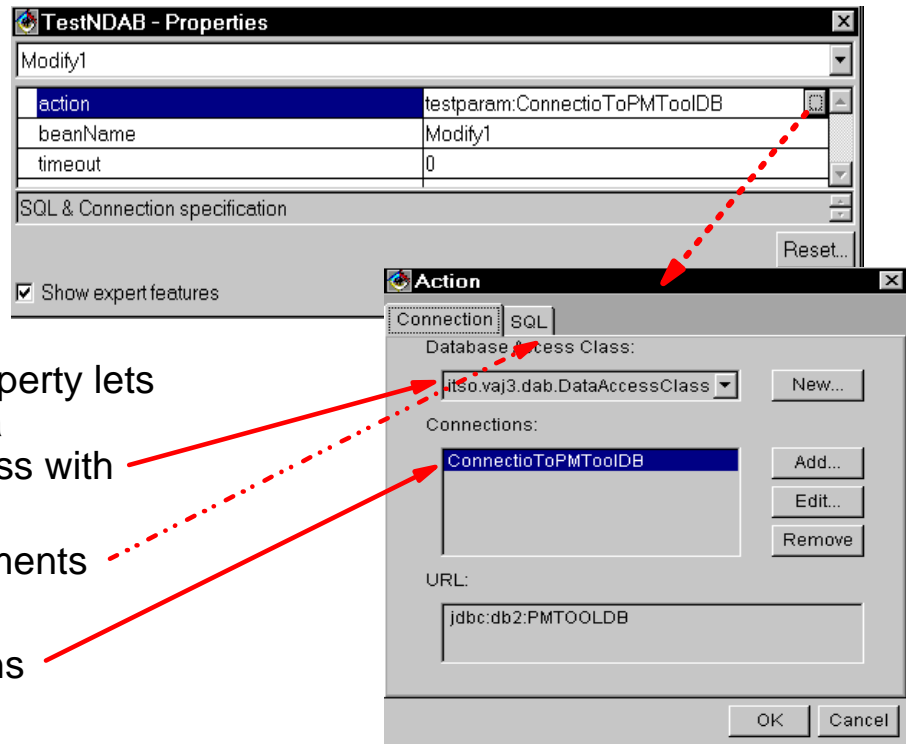


Figure 3-7. Editing the Modify Bean Properties

The MODIFY bean is tailored through its properties.

The **action** property specifies the database connection and the SQL statement. Both are generated as methods into a database access class (same as with the Select bean). You can use a class created previously with a Select bean.

The connection specifies the JDBC driver and database.

The SQL statement is built using a SmartGuide.

## Connection Alias Definition



- The connection alias specifies the database connection characteristics for the Modify bean
- A method in the database access class is generated
- You know this from the Select bean

The screenshot shows a dialog box titled "Connection Alias Definition" with two tabs: "Basic" and "Advanced". The "Basic" tab is selected. It contains the following fields and options:

- Connection Name: [Empty text box]
- URL: [jdbc:db2:sample]
- JDBC Driver Choices: [COM.ibm.db2.jdbc.app.DB2Driver (dropdown menu)]
- JDBC Driver Input: [Empty text box]
- Connection Properties: [Empty text box]
- ☒ Auto-commit
- ☐ Prompt for logon ID and password before connecting
- User ID: [Empty text box]
- Password: [Empty text box]

At the bottom, there are five buttons: "<Back", "Next>", "Finish", "Cancel", and "Test Connection".

*Figure 3-8. Connection Alias Definition*

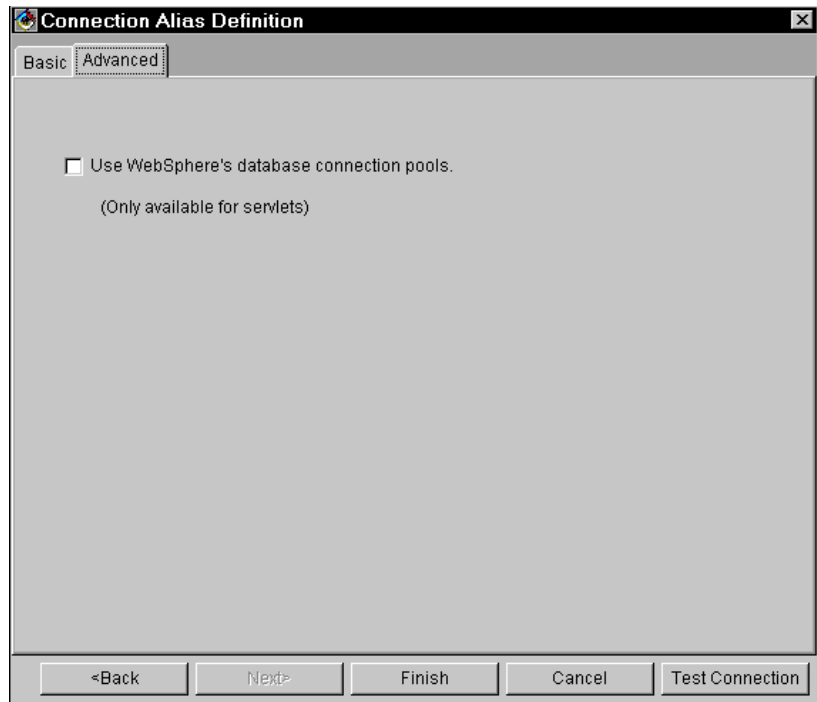
The connection specifies the JDBC driver and database, exactly the same as with the Select bean.



## Using Connection Pools



- Click on the Advanced tab if you want to use **WebSphere's connection pools**
- This is only possible for servlets!
- Also available for the Select bean



*Figure 3-9. Using Connection Pools*

Version 3 provides a new option to use WebSphere connection pools instead of the Data Access Bean connections.

This option is very well suited for servlets, running in WebSphere.

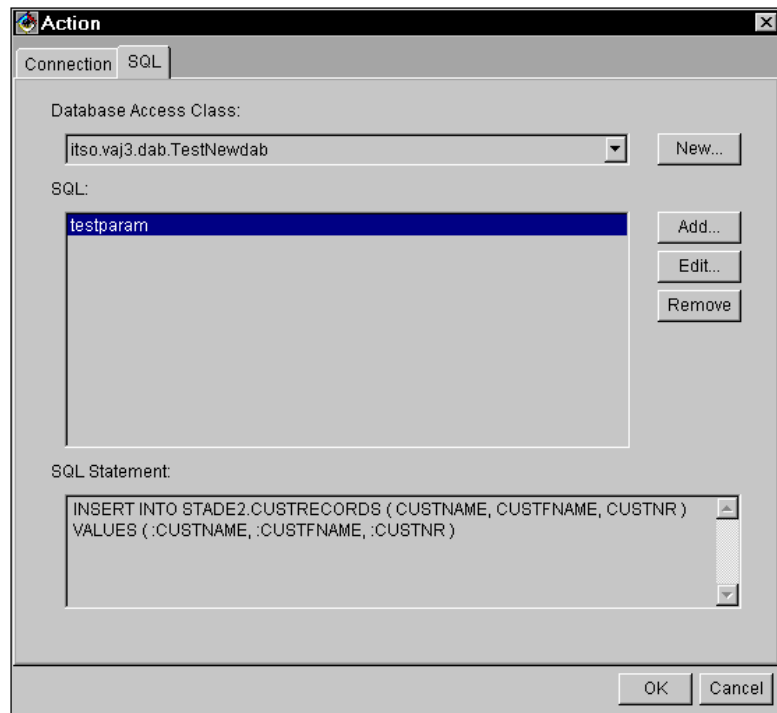
This options also applies to the Select bean.

## SQL Statement Definition



### SQL statement for the Modify bean:

- You can enter the SQL statement manually (an editor is provided)
- You can use the SQL Assist SmartGuide, to help you visually compose the SQL statement.



VA Java V3 - Data Access Beans

© 1999 IBM Corporation

99SWB402UW

Figure 3-10. SQL Statement Definition

The SQL statement is normally built using a SmartGuide, but you can also enter it by hand using an editor. This might be useful for non-standard statements, such as DDL.

The Modify bean can issue any SQL statement.

## SQLAssist SmartGuide



- Allows you quick development of SQL code without being a SQL expert
- The SQLAssist SmartGuide for the Modify bean can only create insert, update and delete statements

SQL Assist SmartGuide

Tables | Insert | SQL

Select an SQL statement type and table(s).

Statement type:

☒ Insert ☐ Update ☐ Delete

Select Table:

Table name	Description
<input checked="" type="checkbox"/> STADE2.CUSTRECORDS	
<input type="checkbox"/> STADE2.PMRECORD	

View schema(s)... Filter table(s)...

< Back Next > Finish Cancel

VA Java V3 - Data Access Beans

© 1999 IBM Corporation

99SWB402UW

Figure 3-11. SQLAssist SmartGuide

The SQL Assist SmartGuide guides you in building an SQL statement for updating the database.

On the first page you select the type of statement and the table to be updated.

## Parameterized SQL Statements



### The Modify bean support the use of parameterized SQL statements

- A parameterized SQL statement contains one or more parameters or variables whose value can be changed as your program runs
  - SELECT \* FROM STAFF WHERE DEPT = :deptNo
  - Program can change the value for :deptNo each time the statement runs
- The Modify bean has methods to set and get the values of parameters
- Each parameter for these beans has a name and can be accessed by name or number
  - INSERT INTO STADE2.CUSTRECORDS ( CUSTNAME, CUSTFNAME, CUSTNR ) VALUES ( :CUSTNAME, :CUSTFNAME, :CUSTNR )  
has three parameters
- Each parameter can be accessed by its name (CUSTNAME,...) or by its ordinal number (1,2,...).

*Figure 3-12. Parameterized SQL Statements*

You can use host variables within the SQL statements.

The SmartGuide generates properties for the parameters. These properties are set before execution using visual construction or method calls. Parameters can be access by number or by name.

## Using the Modify Bean



### Drop the Modify bean to the free form surface

- Set up the connection classes and SQL statements
- With the SQL Assist SmartGuide, VA Java generates two bound properties for each parameter:
  - the parameter in its specified data type
  - a string representation of the parameter

### Connect Modify bean properties to properties of interface components to set parameter values for the SQL statement

- Connect a JButton actionPerformed event to the Parm\_PARAMETERNAME\_String method of the Modify bean
- Make a connection from a JTextField text property to the connection just created and connect the parameter as value

### Provide an execute trigger for the Modify bean:

- connect a JButton actionPerformed event to the execute method of the Modify bean

*Figure 3-13. Using the Modify Bean*

Use the Modify bean in visual construction:

- ☐ set up connections for parameter values
- ☐ set up a connection to issue the SQL statement (execute method)
- ☐ set up connections to retrieve results (such as number of rows updated)

## ProcedureCall Bean



**Use the ProcedureCall bean for running a stored procedure**

**The properties of the ProcedureCall bean are:**

- procedure property (connect to database, define SQL statement)
- memory management properties
  - the ProcedureCall bean can return many result sets and hold them in a cache
  - define, how many result sets should be in the cache and the number of rows for the result sets in the cache
- bound parameters properties
  - for passing parameters to the SQL statement
  - get/set methods for the parameters are created by the SQLAssist Smart Guide

**The methods provided by the ProcedureCall bean are:**

- The execute method executes the SQL CALL statement
- Methods for setting input parameters values and getting output result values

*Figure 3-14. ProcedureCall Bean*

The ProcedureCall bean can be quite complex. A stored procedure has parameters and it may return more than one result set.

Properties of the ProcedureCall bean allow you to set up parameters and to optimize the caching of the result sets.

## Edit ProcedureCall Bean Properties



- Memory management properties

- Use the **procedure** property for setting up connection specs and the SQL CALL statement

TestNDAB - Properties	
ProcedureCall1	
beanName	ProcedureCall1
distinctTypesEnabled	False
fillCacheOnExecute	True
fillResultCacheOnExecute	True
forceSearchedUpdate	False
lockRows	False
maximumPacketsInCache	0
maximumResultsInCache	0
maximumRows	0
packetSize	1
procedure	
readOnly	False
timeout	0

SQL & Connection specification

☒ Show expert features

Reset...

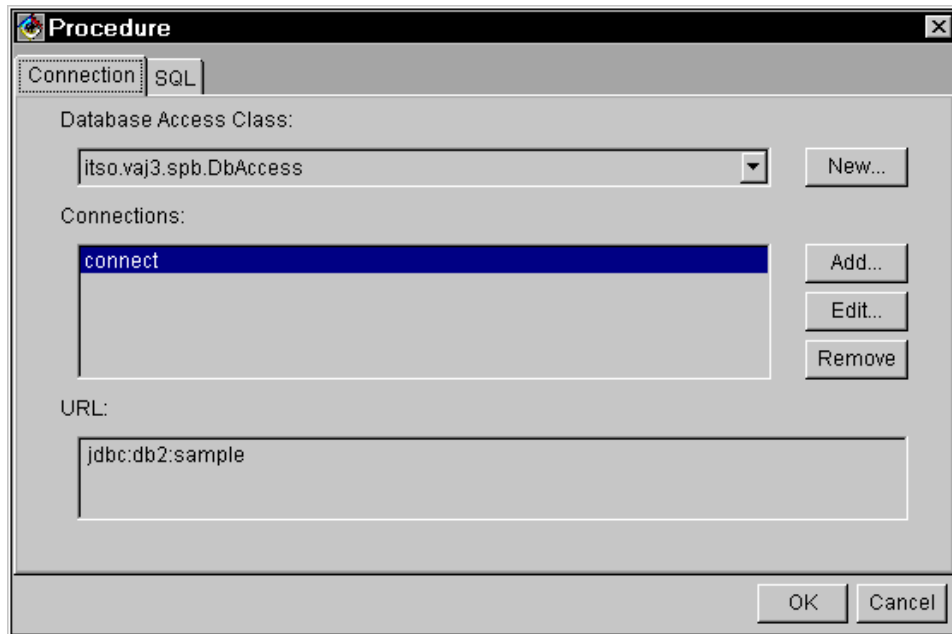
Figure 3-15. Edit ProcedureCall Bean Properties

You gain access to all the properties of the ProcedureCall bean in the Property Sheet.

## Set up the Database Connection



- The same procedure like for the other beans ...



*Figure 3-16. Set up the Database Connection*

The database connection is the same as for other beans.



## Define the SQL Statement



**Write the SQL statement manually, or  
Use the SQLAssist SmartGuide (preferred)**

- First, select the stored procedure, you want to execute

SQL Assist SmartGuide for Stored Procedures

Procedures Parameters Result 1 SQL

Select a stored procedure:

Name	Parameters	Remarks
USERID.DEPTEMPLOYEE	(in WHICHQUERY integer, in DEPTNO character(254))	
USERID.EMPBYJOB	(in JOB character(254))	
USERID.HIGHSALARY	(in SALARY integer)	

View schema(s)

<Back Next> Finish Cancel

VA Java V3 - Data Access Beans

© 1999 IBM Corporation

99SWB402UW

*Figure 3-17. Define the SQL Statement*

Using the SQL Assist SmartGuide you first select which stored procedure to execute.

## Define the SQL Statement (2)



- Add the input and output parameter values:
  - You can use absolute values (static parameters)
  - If no values are provided, automatically parameterized SQL Statements will be used

Name	Mode	SQL type	Input value
JOB	in	character(254)	

VA Java V3 - Data Access Beans

© 1999 IBM Corporation

99SWB402UW

Figure 3-18. Define the SQL Statement (2)

The parameters passed to the stored procedure can be constant values, or (if left empty) parameter properties are generated into the bean.

## Define the SQL Statement (3)



- Create a result set
  - you can create, modify and delete different result sets
- SQL code generated: `CALL STADE2.STP2(:Table)`

SQL Assist SmartGuide for Stored Procedures

Procedures Parameters **Result 1** SQL

Definition of result set 1. Not required if no result set is returned by the stored procedure.

Column	Defined datatype	Treat as datatype
CUSTRECORDS.CUSTNAME	VARCHAR (Type 12)	VARCHAR (Type 12)
CUSTRECORDS.CUSTNR	INTEGER (Type 4)	INTEGER (Type 4)

Define this result set Remove this result set Add another result set

<Back Next> Finish Cancel

VA Java V3 - Data Access Beans

© 1999 IBM Corporation

99SWB402UW

*Figure 3-19. Define the SQL Statement (3)*

On this page you can tailor the result sets returned by the stored procedure.

## Using the ProcedureCall Bean



**You can choose between row-wise and tabular navigation**

### **Row-wise navigation of a result set:**

- use the two bound properties (object + string representation) for each data column in the result set when you used the SQL Assist SmartGuide
  - Example: Make connection between the String representation of a data column in the result set and the text property of a text field. The text field will display the value of the column in the current row of the current result set
- There are also bound properties for the result set
- Using the DBNavigator bean is an easy way to step through result sets
- Set the `currentRow` property of the associated `ProcedureCall` bean to:
  - the first row in the result set
  - the last row in the result set
  - the next row in the result set
  - the previous row in the result set

*Figure 3-20. Using the ProcedureCall Bean*

The bean supports two ways of navigating the results.

In row-wise navigation you step through the rows of the result set (similar to the `Select` bean). You can use a `Navigator` bean to trigger the actions on the result set.

## Using the ProcedureCall Bean (2)



### Tabular display and navigation of a result set:

- property-to-property connection between the *this* property of a Select bean or ProcedureCall bean and the *model* property of a JTable
  - bean methods operate on the current row of the current result set
- To use the JTable to set the *currentRow* property of a ProcedureCall bean make a property-to-property connection between the *selectedRow* property (JTable) and the *currentRow* (ProcedureCall bean)
  - Because the *selectedRow* property is not bound, you will also need to specify an event (*mouseClicked*), to trigger the propagation of the *selectedRow* value to the *currentRow* value
    - This is necessary to insure that methods which operate on the current row (such as *updateRow* and *deleteRow*) function as expected
- In addition, if you will be using methods of the ProcedureCall bean that change its *currentRow* property, such as *deleteRow*, you must trigger the JTable to reflect these changes
  - make an event-to-method connection between the *currentRow* event of your ProcedureCall bean and the *setRowSelectionInterval* method of the JTable

Figure 3-21. Using the ProcedureCall Bean (2)

In tabular navigation you use a Swing JTable that is connected to the ProcedureCall bean.

Using the table you can navigate within the result set. The visual connections have to be setup so that two-way notification is supported.

## Selector Beans



### The Selector beans provide methods for navigating result sets

- All Selector bean have in common:
  - You should use Selector beans to navigate result sets returned by a Select or ProcedureCall bean
  - The Selector beans are working on a table model
  - Use the Selector beans to process a subset of the result set
  - Selector beans have properties you can use to define the subset of data you want to work with
  - They all have data access properties, that allow access to their source data converted to a specified data type
- The differences are:
  - **CellSelector** - provides a single value view of data
  - **CellRangeSelector** - provides a two-dimensional view of data
  - **RowSelector** - provides a row view of data
  - **ColumnSelector** - provides a column view of data

*Figure 3-22. Selector Beans*

The 4 selector beans give access to subsets of a result set. You can access:

- ☐ a single cell value
- ☐ a cell range (a range of rows and columns)
- ☐ a row
- ☐ a column

## CellSelector Bean



The CellSelector bean provides a single value view of data

- **columnName**
  - specifies the name of the selected column from the table model
  - not case sensitive
- **columnNumber**
  - used if columnName is blank
- **indexFromOne**
  - set to true if the CellSelector bean is connected to a model that indexes from one
- **notificationType**
  - specifies which data access property will be notified
- **rowNumber**
  - index of the row to be selected from the source TableModel

Property	Value
beanName	CellSelector1
columnName	
columnNumber	0
indexFromOne	False
notificationType	No notification
rowNumber	0

beanName

Reset...

☒ Show expert features

Figure 3-23. CellSelector Bean

The CellSelector works on a single cell in the result set.

You specify the column (by name or number) and the row (by number).

## CellRangeSelector Bean



**This bean provides a two-dimensional view of data**

- includeColumnNames
  - whether the column name is to be included as the first element of column data (only has effect when column data are retrieved as String values)
- invertData
  - rows \* columns -> columns \* rows
- maximumColumns  
maximumRows
  - select NO\_MAXIMUM means that the last row in the selection should be the last row from the connected TableModel
- startColumnNumber  
startRowNumber

TestNDAB - Properties	
CellRangeSelector1	
beanName	CellRangeSelector1
includeColumnNames	False
indexFromOne	False
invertData	False
maximumColumns	No maximum
maximumRows	No maximum
notificationType	No notification
startColumnNumber	0
startRowNumber	0
vectorContentType	Native
beanName	
<input checked="" type="checkbox"/> Show expert features	
Reset...	

*Figure 3-24. CellRangeSelector Bean*

The CellRangeSelector bean accesses a subset table within the retrieved result table.

You specify the range by giving starting row/column numbers, and the number of rows/columns.

Column names may be included in the output, and the output can be reversed (columns <=> rows).



## RowSelector Bean



### The RowSelector bean provides a row view of data

- vectorContentType
  - data type that should be used when populating vectors in response to a vector property query
  - Some beans may require data to be fed to them in a vector, and may further expect the elements in the vector to be of a specific type
  - You can also set the notification type for propertyChange events to a certain datatype (default is no notification)
- All other properties are similar to the other Selector beans

Property	Value
beanName	RowSelector1
indexFromOne	False
maximumColumns	No maximum
notificationType	No notification
rowNumber	0
startColumnNumber	0
vectorContentType	Native

beanName

☒ Show expert features

Reset...

Figure 3-25. RowSelector Bean

The RowSelector select one row of the result set.

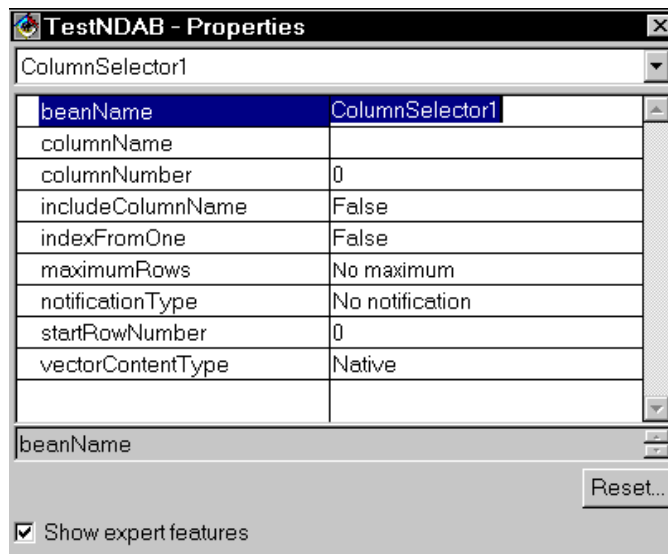
The row values can be converted into a Vector.

## ColumnSelector Bean



**The ColumnSelector provides a single column view of data**

- All properties have the same behavior against columns like the properties from the RowSelector bean against rows



*Figure 3-26. ColumnSelector Bean*

The ColumnSelector select one column of the result set.

The column values can be converted into a Vector.

## Using the Selector Beans



### Example of using a ColumnSelector bean

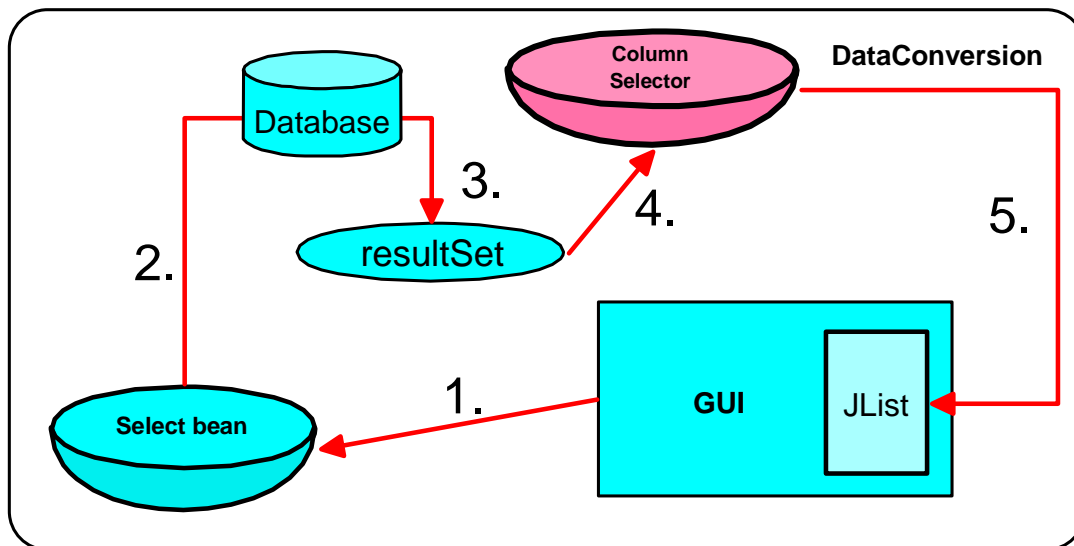


Figure 3-27. Using the Selector Beans

The selector beans are used together with a Select bean (or ProcedureCall bean).

You can select a subset of the result set and display it in some desired format. For example you could display the key values of a result set for the user to select, and then display the details of the selected row.

## Summary



### **Remember, in VA Java 3.0 you still have the**

- Select bean, for creating and executing SQL statements
- The Navigator bean for easy navigation through rows, commit and other RDBMS functions

### **What we have learned about the new beans?**

- The new Modify bean lets you easily insert, update and delete data
- With the new ProcedureCall bean, you can execute stored procedures, access in/out parameters and retrieve multiple result sets
- You will get help by the SQLAssist SmartGuide
- With the four Selector beans, you can manage result sets based on a single cell, cell range, row, or column

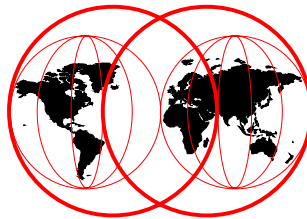
*Figure 3-28. Summary*

The enhancements of the Data Access Beans make programming of relational database access simpler.

# **4 VA Java Version 3 WebSphere Integration**

**VisualAge for Java Version 3**

WebSphere Integration



© 1999 IBM Corporation

99SWB402UW

## Objectives



### WebSphere Application Server (WAS) Overview

#### Using WebSphere Test Environment

- Servlet Application
- JSP Execution Monitor
- XML Applications
- EJB Applications (Enterprise edition only)
- RMI\_IIOP Overview

#### Deployment

#### Testing Environment Additional Configuration

*Figure 4-2. Objectives*

WebSphere Application Server provides the main integrated testing environment for servlets, JSP, XML and EJB applications in VisualAge for Java.

# WebSphere AppServer Overview



## Definition

- Application Web server used to manage high-performance Web sites, including client/server and multi-tier applications based on servlets, server-side script (JavaServer Pages), XML, JavaBeans, and Enterprise JavaBeans

## Advantages

- Application portability, full Java platform
- Excellent browser and server communication
- Facilitates server database communication using Connection Manager
- User friendly administration interface
- Robust deployment environment

*Figure 4-3. WebSphere Application Server Overview*

WebSphere Application Server Version 3 and VisualAge for Java Version 3 are synchronized to work together for development of server-side code.

# WebSphere Test Environment



## Servlet Development

- Setting up Environment
- Create simple servlet
- Testing and Debugging
- Deployment

## JSP Development

- Setting up Environment
- Developing Tools
- Testing and Debugging
- Deployment

## XML Testing Environment

- Setting up Environment
- Testing

## EJB Applications (Enterprise edition only)

- Setting up Environment
- Testing and Debugging

**WebSphere Test Environment can serve  
HTML, servlet, JSP, XML, EJB**

*Figure 4-4. WebSphere Test Environment*

The WebSphere Test Environment provides facilities to run (and debug)

- ☐ Servlets
- ☐ JavaServer Pages
- ☐ XML applications
- ☐ Enterprise JavaBean applications

The Web server provides with this environment can also serve HTML files.



## WebSphere Test Env. Directories



Where to put the files ?

**d:\IBM\Java\ide\project\_resources\**

IBM WebSphere Test Environment\hosts\default\_host\default\_app\web\

- HTML, JSP, XML
- can have any subdirectory structure:
  - ▶ http://localhost:8080/Mydir/Test.html

**Your Project ...\**

- resource files
- servlet control file (.servlet) of PageListServlet
  - ▶ if servlet is itso.vaj3.servlet.Test  
create subdirectories: itso\vaj3\servlet\ for Test.servlet file

*Figure 4-5. WebSphere Test Env. Directories*

The WebSphere provided Web server must find the files and resources.

The directory for HTML, JSP, XML files is in the WebSphere Test Environment project.

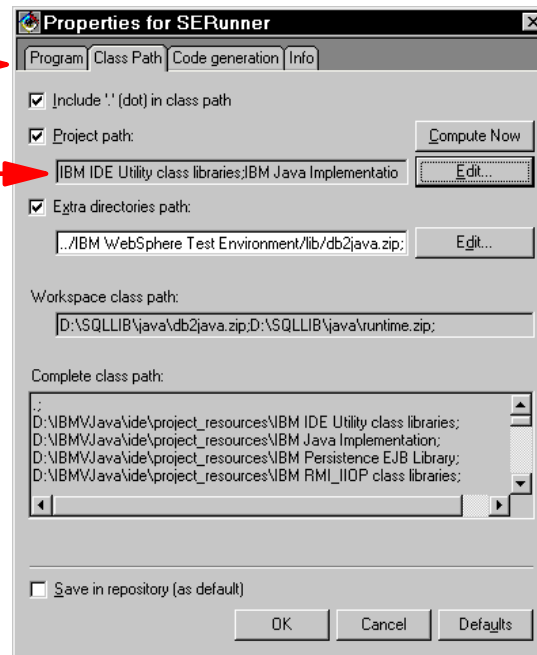
Resource files, such as the servlet control file for PageList servlets must be put into the own project.

## Starting the WebSphere Test Env.



### SERunner class in com.ibm.servlet

- must set class path before starting
  - Run -> Check Class Path
- select all required projects
  - not easy to know sometimes
  - can select "all"



### Run SERunner, or

**Workspace -> Tools -> NEW  
Launch WebSphere Test Env**

- check Console and wait for  
Hostname bindings:  
[hostname-binding]:  
hostname=localhost:8080----> ....

Figure 4-6. Starting the WebSphere Test Env.

The SERunner class is the Web server that must be started.

The class path must be configured to include all the required Java classes. This is not always easy because your code may code other classes (database, Swing, etc). One easy option is to select all projects.

You can run the SERunner class, or you can use the new option:

*Workspace -> Tools -> Launch WebSphere test Environment*

In Version 3 the Test Environment starts faster than in Version 2. The messages in the Console window are different!

# Servlet Development (1)



## Setting up Servlet Development Environment

- Requires Features
  - IBM WebSphere Test Environment 3.0
  - Servlet IDE Utility Library 3.0
  - Servlet Builder 3.0 (Enterprise Edition) for visual servlets

## Create a servlet

- **HTTP Servlets** use  
`javax.servlet.http.HttpServlet` (Servlet API Classes 2.1)  
(subclass of `javax.servlet.GenericServlet`)
- **Visual Servlets** use  
`com.ibm.ivj.servlet.http.VisualServlet` (Servlet Builder 3.0)
  - Can use **Quick Start -> Servlet** option to create **VisualServlet**
- **Page List Servlets (that call JSPs)** use  
`com.ibm.servlet.PageListServlet`  
(subclass of `javax.servlet.http.HttpServlet`)

VA Java V3 - WebSphere Integration

© 1999 IBM Corporation

99SWB402UW

Figure 4-7. Servlet Development (1)

For servlets you can subclass a number of predefined servlet classes:

- ☐ `HttpServlet` provided by Sun in the servlet API
- ☐ `VisualServlet` provided by IBM with the Servlet Builder
- ☐ `PageListServlet` provided by IBM with WebSphere for servlets that call JSPs

## Servlet Development (2)



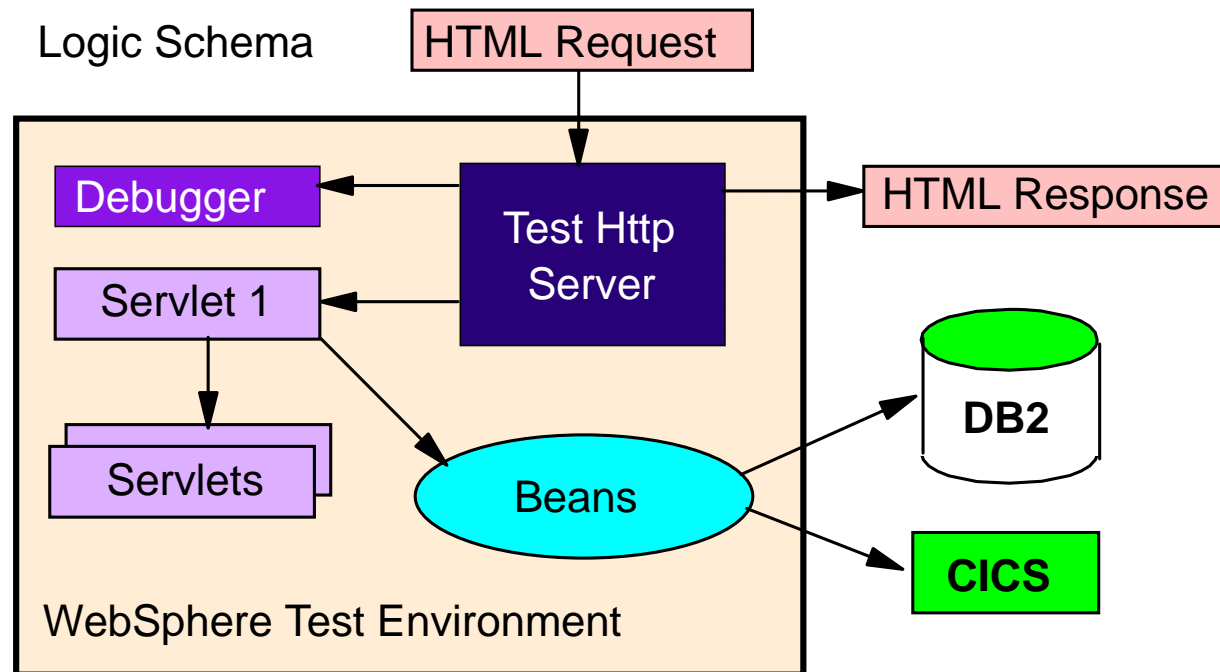
### Testing and Debugging

- Add all projects required for successful operation to
  - com.ibm.servlet.SERunner class path
- Set your breakpoints
- Use servlet launcher feature
  - Select the servlet you want to test and select  
**Tools -> Servlet launcher -> Launch**
- or
  - Start **WebSphere Test Environment**
- Use browser to call servlet
  - <http://localhost:8080/servlet/mypackage.myServlet>
- **Note:** When testing VisualServlet just click Run button

*Figure 4-8. Servlet Development (2)*

You can launch servlets through the Tools menu. This starts the Web server (SERunner) if not already up and running.

## Servlet Development (3)



VA Java V3 - WebSphere Integration

© 1999 IBM Corporation

99SWB402UW

Figure 4-9. Servlet Development (3)

Servlets can use JavaBeans to access enterprise resources (database, transactions, ...).

Servlets can pass control to other servlets or JSPs.

## Servlet Development (4)



### Deployment for WebSphere

- Export all classes needed for a success execution of the servlet
- Put the classes in the application class path
  - In your application environment set
  - servlet name
  - servlet code
  - servlet URI ->  
path appended to the application URI to invoke the servlet
- Sometimes difficult to know all the packages that are required
- Copy the VA Java JAR files  
(in IBM\Java\leab\runtime30 and runtime20)  
to WebSphere and add to class path

*Figure 4-10. Servlet Development (4)*

To deploy servlets in WebSphere it is mandatory to have all classes moved to the WebSphere execution environment.

This is not always easy. In many cases you must copy JAR files provided by VA Java to WebSphere. These files contains classes from:

- ☐ Servlet Builder
- ☐ Persistence Builder
- ☐ Data Access Beans
- ☐ Enterprise transaction systems
- ☐ ...
- ☐ Swing

# JavaServer Pages Development (1)



## Setting up Servlet Development Environment

- Requires Features
  - IBM WebSphere Test Environment
  - IBM JSP Execution Monitor
  - IBM Servlet IDE Utility class libraries

## Version support for JSP 0.91 and JSP 1.0

- tailor file **default\_app.webapp**
  - in \project\_resources\..WebSphere..\hosts\default\_host\default\_app\servlets
  - com.ibm.ivj.jsp.debugger.pagecompile.IBMPageCompileServlet (0.91)
  - com.ibm.ivj.jsp.runtime.JspDebugServlet (1.0)

## Development tools

- WebSphere Studio

## Testing and Debugging

- JSP Execution Monitor
  - Tool provided by VA Java that shows JSP source, generated Java code, and HTML output as it is generated
  - allows setting of breakpoints

VA Java V3 - WebSphere Integration

© 1999 IBM Corporation

99SWB402UW

Figure 4-11. JavaServer Pages Development (1)

JavaServer Pages provide a modern way of separating business logic from presentation.

The business logic is encapsulated into servlets and beans, the presentations id performed in JSPs. The JSP can access data prepared in JavaBeans.

JSPs are compiled into servlet source code. This Java code should not be modified, but it can be used to debug the whole application. A JSP is recompiled when the source is changed.

In VA Java a JSP is recompiled each time after the SERunner Web server has been stopped.

JSPs are coded like HTML, with some extra tags to access JavaBeans. They can also contain Java source code, but that should be limited.

## JavaServer Pages Development (2)



### To use JSP Execution Monitor you have to enable it

- In workspace select **Tools -> JSP Execution Monitor**
- Select enabled monitoring JSP execution and press OK
- If you change the port, it must be between 1024 and 66536

### Then use your browser to call JSP page and you would see the generate code

- Requires WebSphere Test Environment running
- URL is case-sensitive ->URL has to match the root directory structure

### Summary of execution

- Step -> Execute JSP code block by block
- Run -> Executes the JSP source code up to the set breakpoint
- Fast Forward -> Executes the JSP source code *without* stepping
- Terminate -> Stops stepping through the JSP code

Figure 4-12. JavaServer Pages Development (2)

VA Java provides a JSP Execution Monitor that helps with the debugging of the JSP code. The monitor shows the source code, the generated Java code, and the HTML output that is produced by the JSP.



## JavaServer Pages Development (3)



### Retrieve Syntax Error Information

- Specify where the syntax error occurs
  - Errors are displayed in the status line
- Disabled by default
  - in workspace select **Tools -> JSP Execution Monitor**
  - Select enabled monitoring JSP execution and press OK

### JSP Debugging

- JSP Execution Monitor is enabled
  - your code will be in **JSP Page Compile Generated Code** project under pagecompile.\_directoryname\_debug package
- JSP Execution Monitor is disabled
  - your code will be in JSP Page Compile Generated Code project under pagecompile.\_directoryname package

*Figure 4-13. JavaServer Pages Development (3)*

The compiled Java code from JSPs is in a special project called JSP Page Compile Generated Code.

Different packages are used for code compiled with the monitor active or inactive.

## JavaServer Pages Development (4)



### Deployment

- In WebSphere create a new application (Basic)
  - specify application document root URL
  - Add an instance of the JSP processor servlet to the application
  - Place JSP files in application document root
  - Add a pass rule to the Web server to enable serving JSPs and HTML from the new application document root
  - Associates beans or classes should be in application classpath, be sure to export all classes associates with those beans
- If a PageList servlet is used, must also deploy the servlet control file (.servlet), together with the class file

*Figure 4-14. JavaServer Pages Development (4)*

JSPs are deployed like HTML.

JSPs called from a servlet that is a subclass of PageListServlet are named in a servlet control file that must accompany the class code. In the control file all JSPs that can be called are named.

# XML Application Testing



## Setting up XML Testing Environment

- Requires Features
  - IBM WebSphere Test Environment 3.0
  - IBM XML Parser for Java 2.09

## Testing and Debug

- Place XML and DTD file in directory
  - ../WebSphere Test Environment/hosts/default\_host/default\_app/web/
- Develop an application that calls the file using URL
- Launch WebSphere Test Environment and run the application

*Figure 4-15. XML Application Testing*

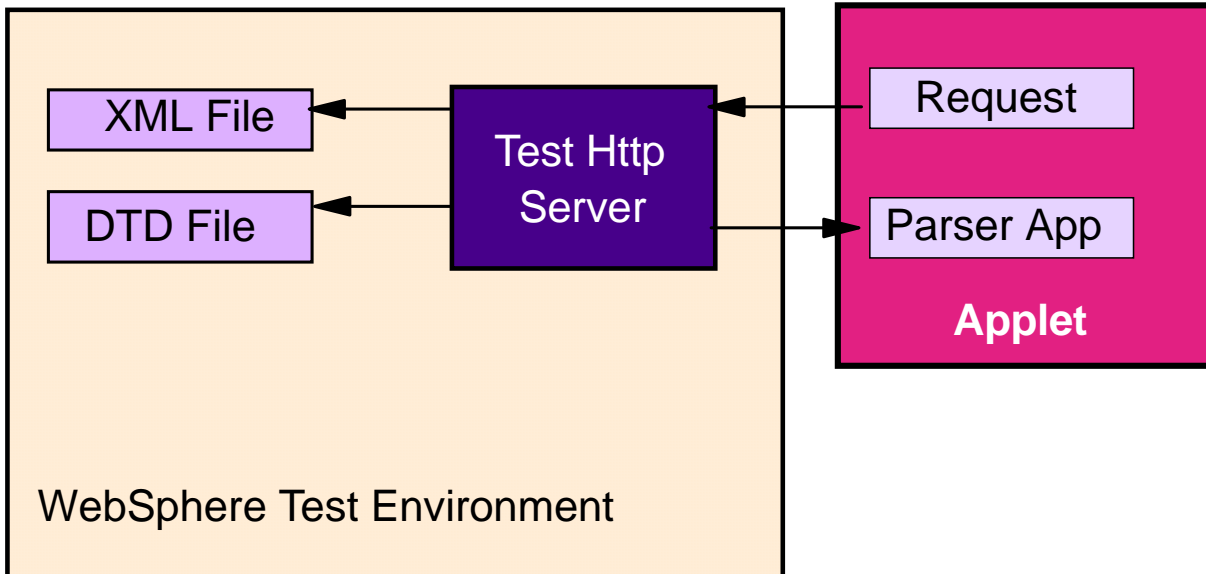
The WebSphere test Environment project also comes with the XML Parser for Java.

With this parser we can develop XML applications. The parser can be used to analyze the content of an XML file and format it into something else.

## XML Logic Schema



### Logic Schema



VA Java V3 - WebSphere Integration

© 1999 IBM Corporation

99SWB402UW

Figure 4-16. XML Logic Schema

The DTD file describes the language (tags) used in the XML file.

The XML file is the content, following the structure prescribed by the DTD file.

The parser analyzes the XML file and provides the content to an application in a different formats. It can create a tree of Java objects (DOM tree) or it can just pass the content item by item to an application class.

## EJB Application Testing (1)



### Setting up EJB Testing Environment

- Requires features
  - IBM EJB Development Environment 3.0
  - IBM WebSphere Test Environment 3.0

### Testing steps

- To understand this section fully you should be familiar with the EJB Development Environment provided by VA Java
- Develop the EJB
- Generate deployed code
- Add EJB group to server configuration
  - Select EJB group and **Add to -> Server Configuration**

*Figure 4-17. EJB Application Testing (1)*

VA Java is the prime development tool for EJBs. We can develop the EJBs, test them, and deploy then to WebSphere.

## EJB Application Testing (2)



### Server Configuration

- Set Naming Server properties and EJB Server properties
  - Location Service Daemon -> Properties
    - ORB Listener Port (default 9000)
  - Select Persistent Name Server -> Properties
    - Database source - InstantDB or Database URL, JDBC Driver
    - LSD Name - name of the host of Location Service Daemon server
    - LSD Port - ORB Listener port (9000)
    - Trace Level - specify the trace level you want in the console
  - Select EJB Server -> Properties
    - Database configuration (URL, JDBC Driver, User ID, Password)
    - LSD Name and Port, Trace Level
- Start Persistent Name Server and EJB Server
- Generate Test Client code
  - Select an enterprise bean -> Generate -> Test Client
- Run Generated Test Client to verify EJB

*Figure 4-18. EJB Application Testing (2)*

In the Server Configuration we specify the database to be used for the persistent data.

We start the name service and the EJB server.

VA Java can also generate a Test client that can be used for initial verification of the functions of the EJB.

## EJB Application Testing (3)



### Test Client

- Click connect button
- Select the method you want to test and click the button new
- Enter values in the text file and click the button send
- Click done button to close the dialog

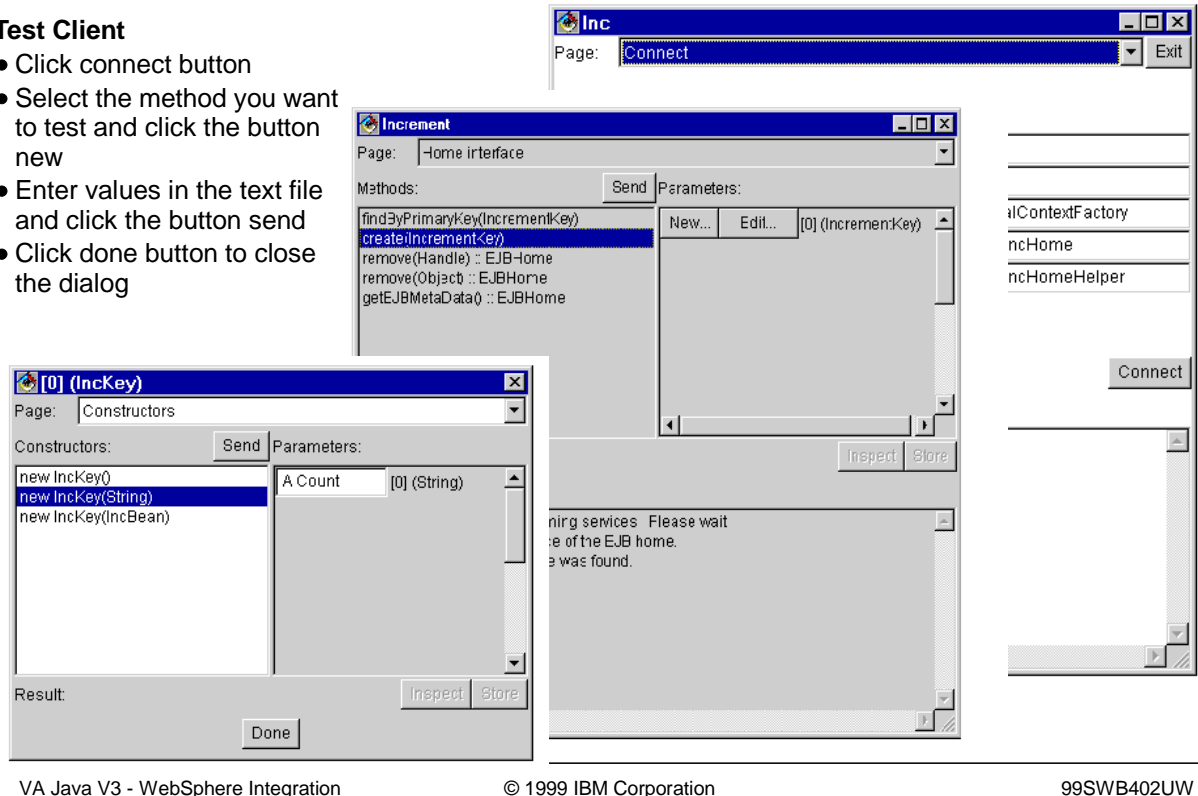


Figure 4-19. EJB Application Testing (3)

With the Test client we can connect to an EJB, retrieve an EJB by key, or create a new one.

Using the remote interface we can then issue the method calls that the EJB provides.

## RMI\_IIOP Overview (1)



### **Remove Method Invocation (RMI) : object distribution technology Java to Java**

- Introduced in JDK 1.1
- Easy to use
- No support for other language

### **Common Object Request Broker Architecture (CORBA): object distribution technology multi-language**

- Open standard
- Multi-vendor
- Not true object-oriented technology

### **Internet Inter-Orb Protocol (IIOP): wire protocol used in CORBA**

VA Java V3 - WebSphere Integration

© 1999 IBM Corporation

99SWB402UW

*Figure 4-20. RMI\_IIOP Overview (1)*

WebSphere and EJBs use the RMI over IIOP protocol for communication.

RMI was provided by Sun in JDK 1.1.1 and was supported in VA Java since Version 1.

CORBA is the open standard API for inter-object communication supporting multiple languages.  
CORBA is used by Component Broker, for example.

IIOP is the protocol used by CORBA.



## RMI\_IIOP Overview (2)



### (RMI) over Internet Inter-Orb protocol (IIOP)

- Mix between the best features of RMI and the best features of CORBA
- no Interface Definition Language (IDL), use RMI subset as interface definition
- allowing to pass serializable Java objects between application components (Object pass by value)
- based on open standard
- ease integration with CORBA support languages

### Reference

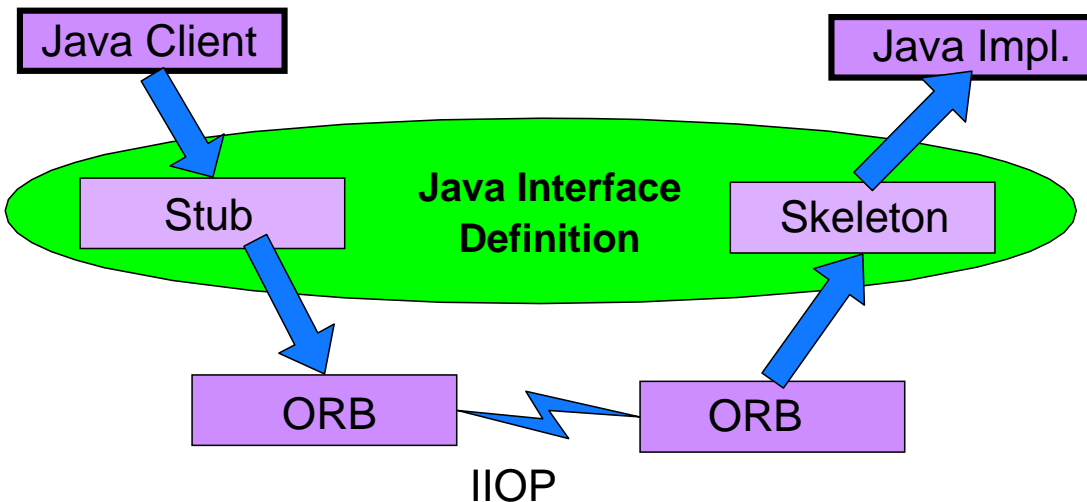
- RMI over IIOP: <http://www.ibm.com/java/jdk/rmi-iiop/>
- RMI\_IIOP Guide: <http://java.sun.com/products/jdk/1.3/docs/guide/rmi-iiop/>
- RMI\_IIOP Home: <http://java.sun.com/products/rmi-iiop/>

*Figure 4-21. RMI\_IIOP Overview (2)*

RMI over IIOP was developed by IBM (with the help and support by Sun).

Basically it supports the RMI protocol from an application point of view, but uses IIOP as the underlying protocol. (RMI had its own private protocol.)

## RMI\_IIOP Architecture Schema



**RMI-IIOP is used by WebSphere and EJBs**

Figure 4-22. RMI\_IIOP Architecture Schema

RMI over IIOP uses the same stub/skeleton approach as RMI.

- ☐ The client talks to a stub
- ☐ The stub serializes the objects and passes them to the server
- ☐ On the server the skeleton receives the serialized objects and makes real objects that are passed to the server class
- ☐ Results are passed back in the opposite direction, again using serialization

## Additional Configuration



### If you want change properties manually:

- JSPDebug.properties -> change the values of JSP Execution monitor Port Num./ Enable Execution Monitor / Enabled Retrieve Syntax Error
- SERunner.properties -> change test environment start properties Port Num./ docRoot path (HTML files)  
// server root (WebSphere Test Environment path)

### Server Launcher Configuration Files

- .../project\_resources/IBM Servlet Builder class libraries  
/com/ibm/ivj/servlet/runner/configuration.properties

### Browser selection option

- Same browser as VAJ use for help Windows -> Options -> Help
- Different browser:  
change browser path in Server Launcher Configuration Files

*Figure 4-23. Additional Configuration*

You can manually change certain configuration options for the WebSphere Test Environment.

## Summary



**WebSphere overview**

**Development, testing, and deployment of servlets and JSP applications**

**Testing XML files and EJB applications**

**Basic knowledge in RMI\_IIOP**

**Configure and launch WebSphere Test Environment**

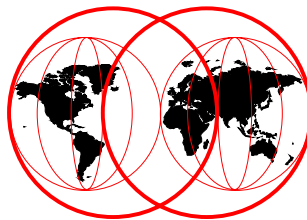
*Figure 4-24. Summary*

VisualAge for Java with the WebSphere Test Environment is the prime development tools for Web server applications.

# **5 VA Java Version 3 Servlet Builder Enhancements**

**VisualAge for Java Version 3**

Servlet Builder



© 1999 IBM Corporation

99SWB402UW

# Objectives



## Overview

- Servlet
- JavaServer Pages (JSP)
- Application Flow

## New Features

- Create Visual Servlet Smart Guide
- JSP Support
- Palette Beans

## Testing Utilities

- Servlet Launcher

*Figure 5-2. Objectives*

The Servlet Builder was introduced in Version 2.

The Enterprise Update provided some enhancements.

Version 3 provides a few additional enhancements that make visual servlets even better.

## Overview



### Servlet

- Sun Definition:  
"Objects which conform to a specific interface that can be plugged into a Java-based server"
- Allow communication between Web browser's pages, Web servers applications and databases (multi-tier applications)
- Improve portability of multi-tier applications

### JSP

- Server-side script
- Allow programmers to embed Java code into HTML pages
- Integrate dynamic contents to static pages
- Access reusable components and session beans
- Compiled when first used or when modified

*Figure 5-3. Overview*

One of the enhancements is that a visual servlet can call a JSP through a bean provided by the Servlet Builder.

## Application Flow



### HTTP Request can be by

- Direct specification of URL
- Selecting HTML link
- Submit Button
- Specifying tag

### Server passes request to visual servlet

- New servlet instance is created to process request (for a visual servlet created with the Servlet Builder)
- Servlets accesses data provided in the HTML form

### Generate an HTML answer

- Return the answer to a browser through the server
- Pass the information to another servlet

*Figure 5-4. Application Flow*

The basic servlet flow is a request from a Web browser through HTML.

The Web server invokes the servlet. For a visual servlet a new instance is created (for an HttpServlet no instance is created, then same object is called).

The servlet access all the data provided in the HTTP call, processes logic possibly using database or transaction access, and finally generates an HTML output that is passed back to the browser.

A servlet can also call another servlet, or call a JSP for output formatting.



## New Features (1)



### Create Servlet SmartGuide

- Is a VisualAge for Java IDE Tool
  - File -> QuickStart -> Servlet -> Create Visual Servlet
- Allow creation of Servlets using different templates
- Quick development servlet environment

### JavaServer Page Support

- Allow user to serve JSP files from a visual servlet
- **JSP Call Wrapper**  
(Requires WebSphere Test Environment for testing)
  - To change the servlet request to a JSP:
    - Set URL path property UriPath = JSP/SimpleJSP/SimpleJSP.jsp
    - then use JSPCallWrapper.forward method to redirect the servlet request to JSP

*Figure 5-5. New Features*

New features of the Servlet Builder:

- ☐ Improved SmartGuide with new templates
- ☐ Call of a JSP through a wrapper bean

The wrapper bean specifies the URL of the JSP file

## New Features (2)



### New Palette Wrapper Beans

- **Variable Wrapper**
  - Use for temporary name/value pairs
  - Pass data between servlets through request block
  - **Pass beans to JSPs**
  - Not saved after servlet execution is complete
- **Attribute Wrapper**
  - Similar to variable wrapper
  - Pass data between servlets through servlet context
  - Data persists and is shared between many servlets
- **Servlet Wrapper**
  - Holds a reference to a visual servlet (servletHandler property)
  - Set isTransferring property to indicate call to other servlet
  - Connect serviceHandler property (of wrapper) to transferToServiceHandler (of calling servlet)

*Figure 5-6. New Features (2)*

New palette beans include wrapper beans for variables, attributes, and servlets.

Variables are temporary for one request only. They are very well suited to pass dynamic information to another servlet or to a JSP.

Attributes belong to a servlet context and persist until deleted. They should not be used for dynamic data, but more to pass some static information from servlet to servlet.

The servlet wrapper specifies another visual servlet to be invoked.

## New Features (3)



### New Palette HTML Bean

- **HtmlDiv** - represent the HTML <DIV> tag
  - With this bean you can separate some HTML areas to use in JavaScript code and create some nice effects

### New Properties

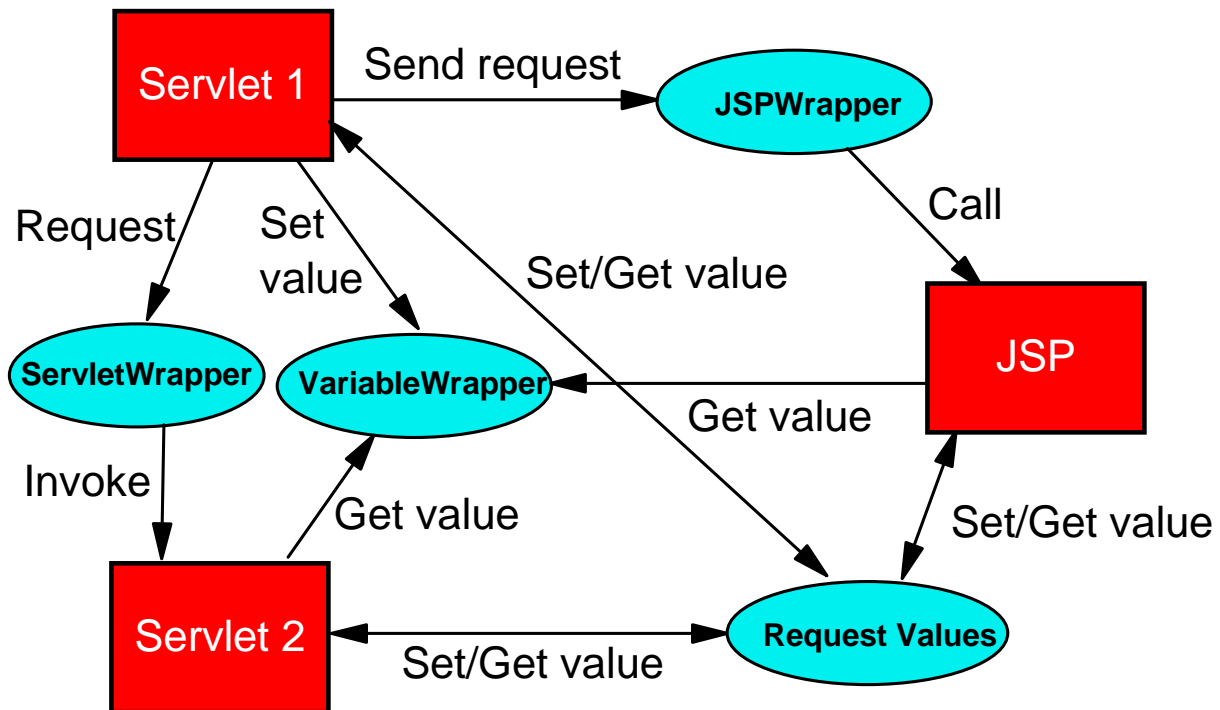
- **HtmlPage>>contentType** - content type of html response
- **HtmlPage>>charSet** - character set for http response
- **HtmlPage>>styleSheet** - now is settable

*Figure 5-7. New Features (3)*

The **HtmlDiv** bean generates the HTML DIV tag.

The **HtmlPage** bean has a few new properties.

## Wrapper Connections



VA Java V3 - Servlet Builder

© 1999 IBM Corporation

99SWB402UW

Figure 5-8. Wrapper Connections

This diagram shows a few ways of passing information between servlets and JSPs.

Wrapper beans are used to encapsulate the data (objects) that is passed from one servlet to another, or to a JSP.

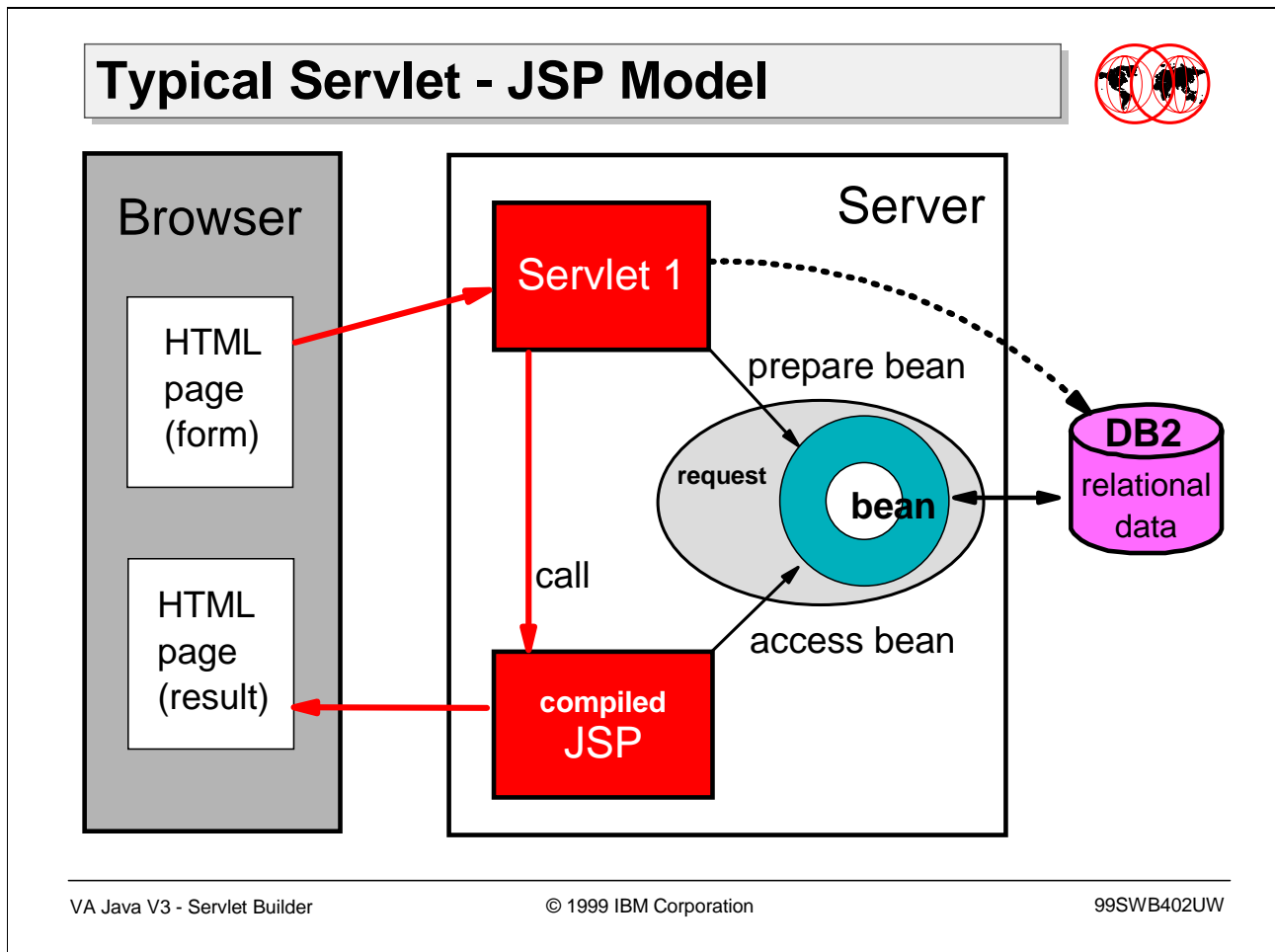


Figure 5-9. Typical Servlet - JSP Model

In the typical servlet-JSP model, a servlet does most of the processing. JavaBeans (or Enterprise JavaBeans) are used to access enterprise data (databases, transactions).

Results are stored in JavaBeans to be accessed from the JSP that prepares the output.

The Appropriate JSP is called for output formatting. Through special tags (<BEAN>) the JSP has access to JavaBeans that have been registered in the request block.

The JSP uses the data from the JavaBeans to dynamically format the HTML output for the browser.

## Servlet Launcher (1)



### VisualAge for Java IDE Tool

- Can start an HTTP Server for test and debug visual servlets
  - Automatically start a Web browser with the selected servlet

### Workspace Projects

- Servlet IDE Utility library 3.0
- Servlet Builder 3.0
- Sun Servlet API 2.1 (JSDK)
- WebSphere Test Environment 3.0

### HTTP Server

- Default **SERunner** at port 8080 (WebSphere Test Environment)
  - best to start before servlet is run
- JSDK Http server (servlet runner)

*Figure 5-10. Servlet Launcher*

The servlet launcher can be used to start a servlet from the IDE, instead of entering a long HTTP request in a browser.

By default the WebSphere Test Environment is used as HTTP server, but you can configure to use the older Web server provided with the JSDK (Servlet Development Kit) from Sun. The JSDK server does not process HTML requests, only servlet requests.

## Servlet Launcher (2)




### Change Configuration

- **Http server**

.../project\_resources/IBM Servlet Builder class libraries  
/com/ibm/ivj/servlet/runner/configuration.properties.

- **HTML Browser**

Window->Options->Help



```
serverPort=8080
serverAddress=127.0.0.1
serverDocBase=.
serverClassPathSuffix=../Sun JSDK class libraries/webserver.jar;.....
serverDirectory=../project_resources/IBM WebSphere Test Environment
serverDirectoryRootProperty=user.home
#browserPath=c:/netscape/communicator/program/netscape.exe
```

Figure 5-11. Servlet Launcher (2)

The servlet launcher can be configured in a properties file.

## Summary



**Review JSP and servlet concepts**

**Build robust servlet application using the new features of Servlet Builder**

**Join JSP and servlets in a visual environment**

**Testing servlets in the development environment**

*Figure 5-12. Summary*

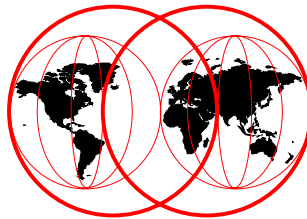
Servlets and JSPs are the most modern way for server-side applications, together with Enterprise JavaBeans for access to enterprise resources.



# **6 VA Java Version 3 XML Parser for Java**

**VisualAge for Java Version 3**

XML Parser for Java



© 1999 IBM Corporation

99SWB402UW

# Objectives



## Extensible Markup Language (XML) Overview

### IBM XML Parser for Java 2.09

- DOM (Document Object Model)
- SAX (Simple API for XML)

### Testing

### Reference sites

### Lotus XSL Processor

*Figure 6-2. Objectives*

This unit provides a basic understanding of XML and the support for XML in VisualAge for Java.

We cover very shortly the two main XML Parsers: the DOM Parser and the SAX Parser.

We also have a short look at the Lotus XSL Processor that translates XML into HTML.

## XML Overview



**XML represent the contextual meaning of the data**

### **Advantages of XML**

- Easy to use and understand
- Can extend XML to create different dialects (Document Type Definition - DTD)
- A lot of the programming is already done for you, just change your DTD
- XML more versatile than HTML
- Represent data as information structure

**The DTD file describes the syntax of the XML document**

**Can translate XML to objects using Document Object Model (DOM)**

**Can analyze XML using the Simple API for XML (SAX)**

*Figure 6-3. XML Overview*

XML: the new standard of moving data between applications.

DTD describes the structure and tags of an XML document.

XML file contains the data, structured according to the DTD.

Some browsers can display XML already (Internet Explorer 5.0).

XML content can be translated into an object structure using a DOM parser.

There is also a simple, event driven, API (SAX).

## IBM XML Parser for Java



### **DOM = Document Object Model: `org.w3c.dom`**

- Package has relevant classes to create an object structure for an XML document
- Documents can be manipulated as objects

### **SAX = Simple API for XML: `org.xml.sax`**

- Package with relevant classes to handle XML files and parse to Java
- Parser
  - main interface allows the user to register handlers for callback, set the location for error reporting, and to start the parse operation
- HandlerBase
  - provides default implementation to handle the XML file
  - programmer can use to start the handler phase
- Standard interface for event-based XML parsing
  - program method is called for every XML tag

*Figure 6-4. IBM XML Parser for Java*

IBM provides an XML parser, originally on the AlphaWorks site, now included in the WebSphere Test Environment in VA Java. (Also included in WebSphere Application Server.)

The XML parser provides packages for DOM and for SAX processing.

## XML Parser Example



### To use IBM XML Parser for Java in VisualAge for Java:

- Add IBM XML Parser for Java Feature
  - or add WebSphere Test Environment

### Example using the SAX Parser:

- Create DTD file
- Create XML file
- Use the **Simple API for XML (SAX)** for process an XML file in Java code
  - Create a class that implements `org.xml.sax.Parser`, or a class that extends `org.xml.sax.HandlerBase`, to control the action that is performed while reading the XML text
- Create the application that calls the parser
- Add IBM XML Parser for Java project to the application class path

*Figure 6-5. XML Parser Example*

The parser must be added to the Workbench.

In this example we use the Simple API for XML (SAX) to analyze an XML file.

## XML File Example



```
<?xml version="1.0"?>

<!DOCTYPE Lottery SYSTEM "winnerNo.dtd">

<Lottery>
  <DrawingInformation>
    <DrawingNo value="189"/>
    <DrawingDate>May 25,1999</DrawingDate>
    <DrawingType>Extraordinary</DrawingType>
    <WinnerNo>
      <InformationNo>
        <No>12345</No>
        <PriceTy>First Price</PriceTy>
      </InformationNo>
    </WinnerNo>
  </DrawingInformation>
</Lottery>
```

Data can be represented as attributes or between start and end tags

Figure 6-6. XML File Example

Here is a sample XML file for a lottery drawing.

In XML it is mandatory to have an end tag for every begin tag.

Tags can be structured, that is, one set of tags inside another tag.

Some tags are like groups, they do not have any data. Others have data.

Note:

- Data can also be specified as attributes:

```
<DrawingNo value="189"></DrawingNo>
```

## DTD File Example



```
<?xml version="1.0"?>
<!ELEMENT Lottery (DrawingInformation)+>
<!ELEMENT DrawingInformation (DrawingNo,DrawingDate,
                               DrawingType,WinnerNo)>
<!ELEMENT DrawingNo EMPTY>
  <!ATTLIST DrawingNo value CDATA #REQUIRED>
<!ELEMENT DrawingDate (#PCDATA)>
<!ELEMENT DrawingType (#PCDATA)>
<!ELEMENT WinnerNo (InformationNo)+>
<!ELEMENT InformationNo (No,PriceTy)>
<!ELEMENT No (#PCDATA)>
<!ELEMENT PriceTy (#PCDATA)>
```

*Figure 6-7. DTD File Example*

This is the sample DTD file for the XML file.

The DTD file describes the structure of the tags. For each tag it describes the tags that are allowed inside or that the tag has associated data.

Think of the DTD as the syntax specification of the XML file.

## Java Handler Code Example



```
import org.xml.sax.HandlerBase;
import org.xml.sax.AttributeList;
public class DrawingHandler extends org.xml.sax.HandlerBase {
    public void startElement(String name, AttributeList atts) {
        System.out.println("Start Element: "+name);
    }
    public void endElement(String name) {
        System.out.println("End Element: "+name);
    }
    public void characters(char[] ch, int start, int lg) {
        System.out.println("Data: "+(new String(ch,start,lg)).trim());
    }
}
```

VA Java V3 - XML

© 1999 IBM Corporation

99SWB402UW

*Figure 6-8. Java Handler Code Example*

This example shows how the `org.xml.sax` package can be used to parse the XML file and to call an application handler class with all the tags.

For each tag the handler class is called in a `startElement`, `endElement`, and `characters` method.

- ❑ The `startElement` method call includes the data attributes associated with the tag:

```
<tag value=xxxxxx>
```

- ❑ The `endElement` method is called when the end tag is encountered:

```
</tag>
```

- ❑ The `characters` method is called with the data that is between the start and end tags:

```
<tag> xxxxxxxxxxxxxxxxx </tag>
```



## Java Application Code Example



```
import org.xml.sax.Parser;
import org.xml.sax.DocumentHandler;
import org.xml.sax.helpers.ParserFactory;
public class MyApp {
    //class used to parse XML file
    static final String parserClass = "com.ibm.xml.parsers.ValidatingSAXParser";
    public static void main (String args []) throws Exception {
        //create an instance of the parser
        Parser parser = ParserFactory.makeParser(parserClass);
        //create instance of the handler class
        DocumentHandler handler = new DrawingHandler();
        parser.setDocumentHandler(handler);
        //perform the parsing for the file passed as parameter
        parser.parse(args[0]);
    }
}
```

*Figure 6-9. Java Application Code Example*

This application code example shows how to set up the parser, how to register the handler class with the parser, and how to start the parsing operation.

## Testing



### Add WebSphere Test Environment Feature

- this adds the XML Parser project

### Insert XML and DTD file into directory

- IBM\Java\ide\project\_resources\WebSphere Test Environment\hosts\default\_host\default\_app\web
- XML file in subdirectory (xml\winnerNo.xml)

### Launch WebSphere Test Environment and call the XML file from the application using localhost URL:

- <http://localhost:8080/xml/winnerNo.xml>

*Figure 6-10. Testing*

To test such an XML application you start the WebSphere Test Environment, prepare the XML and DTD files, and call the XML file from the application code using an URL.

This is more flexible than pointing directly to files.

## Reference Sites



**XML home** <http://www.xml.com/>

### **Documents for beginners**

XML for absolute beginner

<http://www.javaworld.com/javaworld/jw-04-1999/jw-04-xml.html>

**Document Object Model home** <http://www.w3c.org/DOM/>

### **SAX API Reference**

<http://www.megginson.com/SAX/index.html>

### **IBM sites:**

<http://www.ibm.com/xml/>

<http://w3.xml.ibm.com/>

*Figure 6-11. Reference Sites*

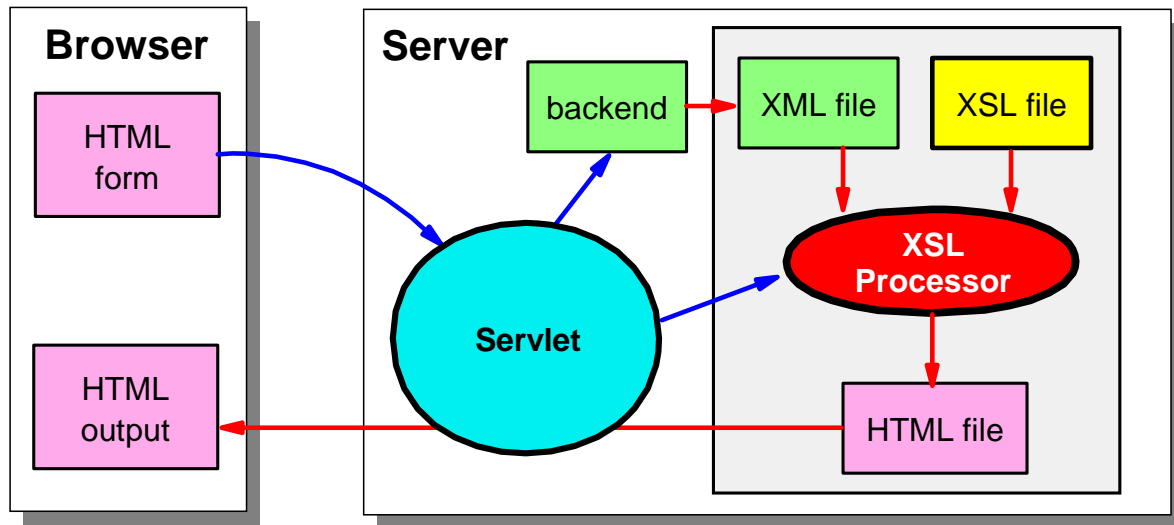
For more information on XML, look on the Internet; there is more information than you can ever read!

## Lotus XSL (VA Java Extras)



**Lotus XSL Processor converts an XML file into an HTML file using an XSL file (eXtensible Stylesheet Language)**

- XML tags are formatted into appropriate HTML tags



VA Java V3 - XML

© 1999 IBM Corporation

99SWB402UW

Figure 6-12. Lotus XSL

Another way of dealing with XML files is to translate the file into HTML.

Lotus provides the XSL processor for this purpose. An XSL processor uses an XSL file that describes how each XML tag is translated into HTML. This allows for creative formatting of XML information into HTML output.

## Lotus XSL Example



```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">
  <xsl:output method="xml" indent="yes"/>
  <xsl:template match="Lottery">
    <html> <head> <title> Winner Numbers </title> </head> <body>
      <xsl:apply-templates/>
    </body> </html>
  </xsl:template>

  <xsl:template match="DrawingInformation/DrawingNo">
    <h2> Drawing Number is <xsl:value-of select="@value"/> </h2>
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="DrawingInformation/DrawingDate">
    <ul><li> Drawing Date is <b> <xsl:apply-templates/> </b> </li></ul>
  </xsl:template>

  ....
</xsl:stylesheet>
```

Figure 6-13. Lotus XSL Example

This example shows an extract of an XSL file to format our sample XML file.

For each tag in the XML file, the appropriate HTML tags are emitted to produce an output.

A servlet can call the XSL processor with the XML and XSL file and have the resulting HTML output routed directly to the browser (instead of an output file).

## Summary



### **Overview of XML**

**Know most relevant classes in IBM XML Parser for Java**

**How develop and test applications using XML and Java in a  
VA Java Environment**

**Understand the basic principle of the Lotus XSL Processor**

*Figure 6-14. Summary*

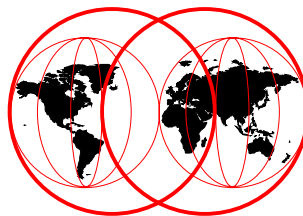
XML is the next big wave.

XML and Java work very well together, and IBM has a great solution with the XML Parser, WebSphere, and VisualAge for Java.

# **7 VA Java Version 3 Persistence Builder Enhancements**

**VisualAge for Java Version 3**

Persistence Builder



© 1999 IBM Corporation

99SWB402UW

## Objectives



### **Review of Persistence Builder**

### **Enhancements to the Persistence Builder**

#### **New functions:**

- Thread support
- Connection pooling
- Browser enhancements

#### **Existing models must be regenerated**

- Domain classes
- Service classes

#### **Check the model, schema, map before regeneration**

- Browser validation has been improved and errors may be found

*Figure 7-2. Objectives*

In this unit we review the Persistence Builder and the improvements that were made for Version 3.



## Persistence Builder Review



### ■ Application Programming

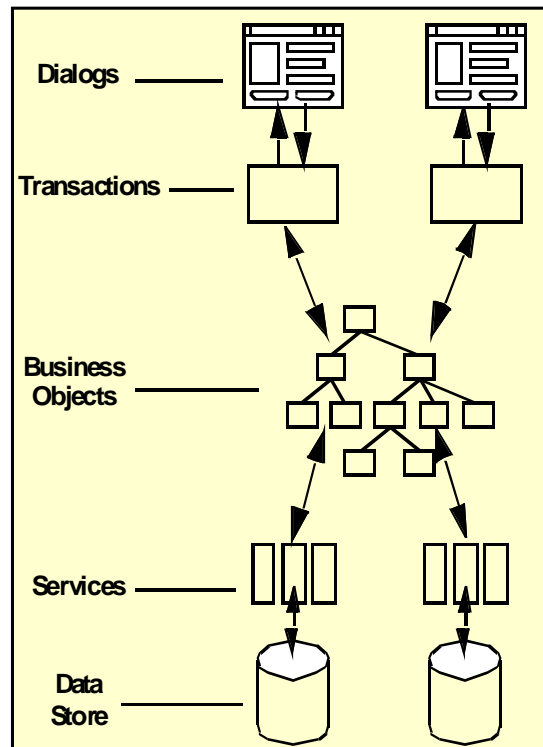
- user interface
- unit of work (transactions)

### ■ Object Modeling

- business objects
- relationships between obj.

### ■ Data Access Programming

- mapping from objects to DB schema (tables)
- services (storage and retrieval)



VA Java V3 - Persistence Builder

© 1999 IBM Corporation

99SWB402UW

Figure 7-3. Persistence Builder Review

The Persistence Builder uses an object model as the central facility. The application works with the object model and uses a transaction framework to deal with concurrency and locking.

The object model is mapped into relational database tables. All the service code to instantiate objects from the database and to update the database are generated by the Persistence Builder.

## Key Elements



### Business Object

- real world, identification, versions (per transaction)
- business layer generated

### Relationships

- between business objects (1:1,1:m,m:m), referential integrity

### Transactions

- unit of work, nested, concurrent, locking (pessimistic, optimistic)

### Mapping

- object - table, foreign keys, inheritance, multiple tables, complex
- service layer generated

### Datastore

- workspace, relational, queries

*Figure 7-4. Key Elements*

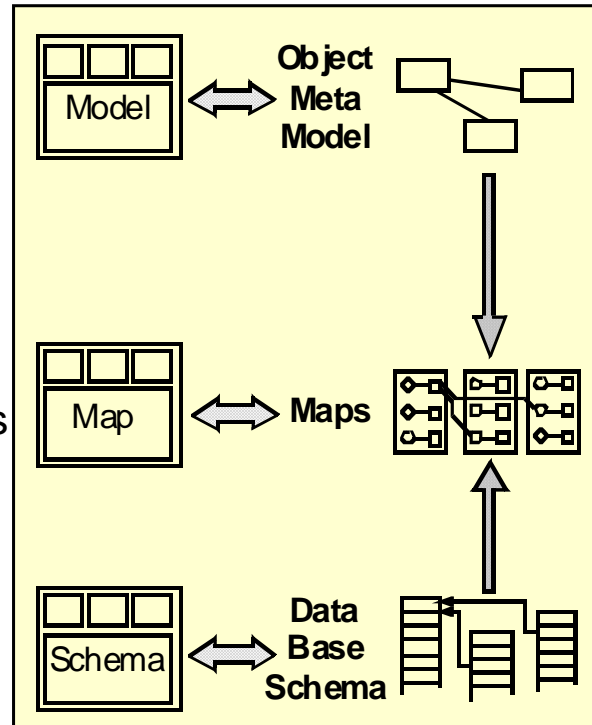
The Persistence Builder works with:

- ☐ Business objects (the application objects)
- ☐ Relationships between business objects are supported with referential constraints
- ☐ Transactions are used for concurrency, locking, commit/restore
- ☐ Maps define how the business objects are mapped into relational tables
- ☐ Datastore: the physical store (usually a relational database)

## Development Environment



- VA Java Workbench
  - **Model Browser** for persistent classes and relationships
    - generate model code
  - **Map Browser** for mapping between objects and tables
    - generate services code
  - **Schema Browser** for logical database schema
    - generate DB tables



VA Java V3 - Persistence Builder

© 1999 IBM Corporation

99SWB402UW

Figure 7-5. Development Environment

The development environment of the Persistence Builder is a set of 3 browsers:

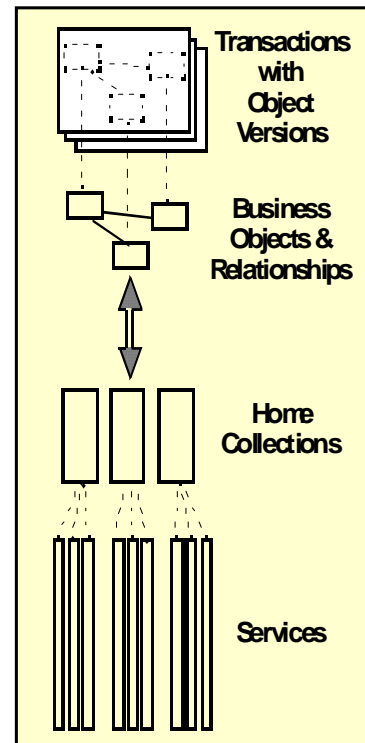
- ☐ Model Browser: deals with the business objects
- ☐ Schema Browser: deals with the database schema (relational tables)
- ☐ Map Browser: maps the business model to the table model

From the browsers the Persistence Builder generated the Java code for the business objects, the Java code for the services (access to the database), and the DDL for the database tables.

## Runtime Environment



- **Transaction management**
  - logical units of work
  - multiple versions of business objects
- **Business object classes**
  - generated from model
- **Home collections are main interface**
  - find, create of business objects
- **Service classes interface with the database**
  - generated from mapping



VA Java V3 - Persistence Builder

© 1999 IBM Corporation

99SWB402UW

Figure 7-6. Runtime Environment

The runtime environment consists of

- Transactions: to handle units of work, concurrency, commit/rollback
- Business Objects: the application data
- Home Collections: used to find and create business objects
- Service classes: to do all the work between the tables and the business objects

## Development Paths



### ■ Forward Engineering

- start with object model
- generate schema and mapping (and tailor)

Forward or Top-Down: Start with the Metamodel

### ■ Backward Engineering

- start with database schema (import)
- generate model and mapping (and tailor)

Backward or Bottom-Up: Start with the Database

### ■ Outside-In or Mapping

- start with object model and existing database schema
- create a tailored mapping

Outside-In: Map the model

to the Database Schema

Figure 7-7. Development Paths

For a complete system you must have a model, a schema, and a map.

You can develop these top-down, bottom-up, or meet in the middle.

## Multithreading (1)



### **Persistence Builder 3.0 supports multithreading**

- V2.0 only supported explicit Transaction suspend and resume
- Made multithreaded programming difficult

**In a multithreaded environment the transactions have to be bound to threads**

### **The two ways for doing the binding:**

- Each thread has an instance variable that holds on to the current transaction within the thread
- Persistence Builder maintains a transaction lookup table which is keyed by the threads

*Figure 7-8. Multithreading (1)*

New in Version 3 is multithreading support.

This allows the creation of real multithreaded applications.

## Multithreading (2)



### Transaction class holds on to the system-wide binding policy

- `Transaction.setBindingPolicy(TransactionBindingPolicy aPolicy)`

### The two predefined policies are:

- **TransactionToContextThreadBindingPolicy**
  - Uses the instance variable approach
  - Each thread has to implement the `ContextThread` interface for accessing the context holding the current transaction
  - default
- **TransactionToThreadBindingPolicy**
  - Uses the lookup-table approach
  - preferable in WebSphere/servlet environment
    - `Transaction.setBindingPolicy(new TransactionToThreadBindingPolicy());`

*Figure 7-9. Multithreading (2)*

How the multithreading support is handled by the application is through the transactions.

There are two possible policies, instance variables, or look-up table.

## Connection Pooling



### **New connection options in services generation:**

- Use specified login ID and password for database connections
- Supply login ID and password at runtime for database connections
- Use database connections from WebSphere connection pool

### **Extensible connection policy classes:**

- VapConnectionPolicy (use PB connections)
- WebSphereConnectionPolicy (use WebSphere connections)

### **When to use Websphere connections:**

- Web applications that incur high volume can avoid connection startup times
- Multiple applications can avoid maximum connection problems by utilizing the same connection scheme

*Figure 7-10. Connection Pooling*

New in Version 3 is connection pooling.

In addition to the Persistence Builder connection you can now choose the WebSphere connection pools.



## Metamodel Validation



**The old browsers allowed the user to create invalid constructs**

**The problems sometimes manifested themselves during code generation, sometimes run time errors occurred**

**Invalid constructs can exist in old metadata:**

- Invalid constructs can result from multiple independent actions on the browsers
- For example, type mismatches

**The browsers have been enhanced to flag these invalid constructs so the user can take action immediately**

*Figure 7-11. Metamodel Validation*

Version 3 provides better validation between model - schema - map.

The effect is that more reliable code is generated. Some old models may have validation errors when imported into version 3.

## Renaming Metamodel Elements



### Renaming Schema elements

### Renaming Model elements

#### Examples:

- Rename table (Tables --> Rename Table)
- Rename column
- Rename foreign key relationship
- Rename class
- Rename attribute

**Renaming allows you to change names without forcing you to delete the element and recreate it as a new element**

*Figure 7-12. Renaming Metamodel Elements*

Version 3 supports more rename facilities. This allows to use more application specific names, even after a model/schema/map has been generated.

## Mapping for EJBs



### EJBs use the same Map and Schema browsers as Persistence Builder

#### There are some differences in how the Map browser is used:

- Datastore Maps pane
  - New EJB Group Map...
  - Generate Services
- Persistent Classes pane
  - Enable pessimistic locking
  - Change default preload path
- Property Maps pane
  - Be part of optimistic predicate

*Figure 7-13. Mapping for EJBs*

When developing EJBs you use the same schema and map browsers.

In the context of EJBs, they have a few different options, but the basic principles are the same. EJBs do support now relationships and inheritance, and the mapping is done using the Persistence Builder tools.

The generated code is, however, different.

## SQL Datatype Framework



### Open to the developer:

- Developer can add new datatypes or Java types to the map as needed
- Developer can change how datatypes map to Java types and vice-versa
- Specify default converters for Top-Down Programming for particular mappings
- Specify default converters for SQL datatypes

### Developer can change virtually anything that is specific to a SQL datatype including:

- SQL Number
- DDL String
- Fixed, Variable, Scaled
- Filler value

*Figure 7-14. SQL Datatype Framework*

Vendors may use different SQL data types for the same data. This can lead to problems.

In Version 3 the datatype mapping has been improved to allow more flexibility.

The Datatype Framework controls how SQL datatypes are mapped to Java types. In version 2.0, this framework is hidden from the users; thus, making it impossible to change.

In version 3.0, this framework has been redesigned to give the developer more control; however, these changes will not be documented. So the knowledge of these changes will only be dispensed as needed. Customers who need this type of flexibility would include customers that have specific object types that they want to use with Persistence Builder or customers that have datatype mapping issues that conflict with the default.

## Summary



**The Persistence Builder has not changed much between VisualAge for Java Version 2 and 3**

**New functions:**

- Thread support
- Connection pooling
- Browser enhancements

**Existing models must be regenerated**

*Figure 7-15. Summary*

The Persistence Builder is basically the same as in Version 2.

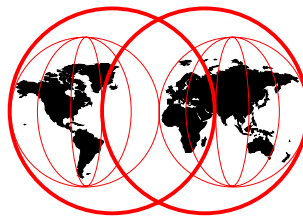
More congruence between the Persistence Builder and Enterprise JavaBeans.



# **8 VA Java Version 3 Enhanced EJB Support**

**VisualAge for Java Version 3**

Enhanced EJB Support



© 1999 IBM Corporation

99SWB402UW

## Objectives



### **Understand the enhancements of EJB Development Environment provided by VA Java Enterprise Edition V3**

**The new EJB Development Environment provides you many new features:**

- EJB Inheritance Support
- Association Support
- Enhancements in Mapping Approach
- Finder Helper Enhancements
- Access Bean

**To understand this section fully:**

- you should be familiar with the EJB Development Environment provided by VA Java Enterprise Edition V2 & Enterprise Update
- you should also be familiar with EJB (Enterprise JavaBeans)

*Figure 8-2. Objectives*

The EJB development facility in VisualAge for JAVa has been greatly enhanced together with better deployment facilities in WebSphere.



## EJB - Enterprise JavaBeans (1)



### EJB is the specification of server-side Java component

- Defined by Sun Microsystems
- Compatible with CORBA

### There are 2 types of EJB

- **Session Bean**

- This EJB executes business logic on behalf of a single client in a server
- For example, you can implement a set of transactions in a session bean

- **Entity Bean**

- This EJB represents persistent business data, such as, customer, account, ...
- There are 2 types of Entity EJBs

- **BMP (Bean-Managed Persistence)** entity bean

- EJB developer must develop persistence layer

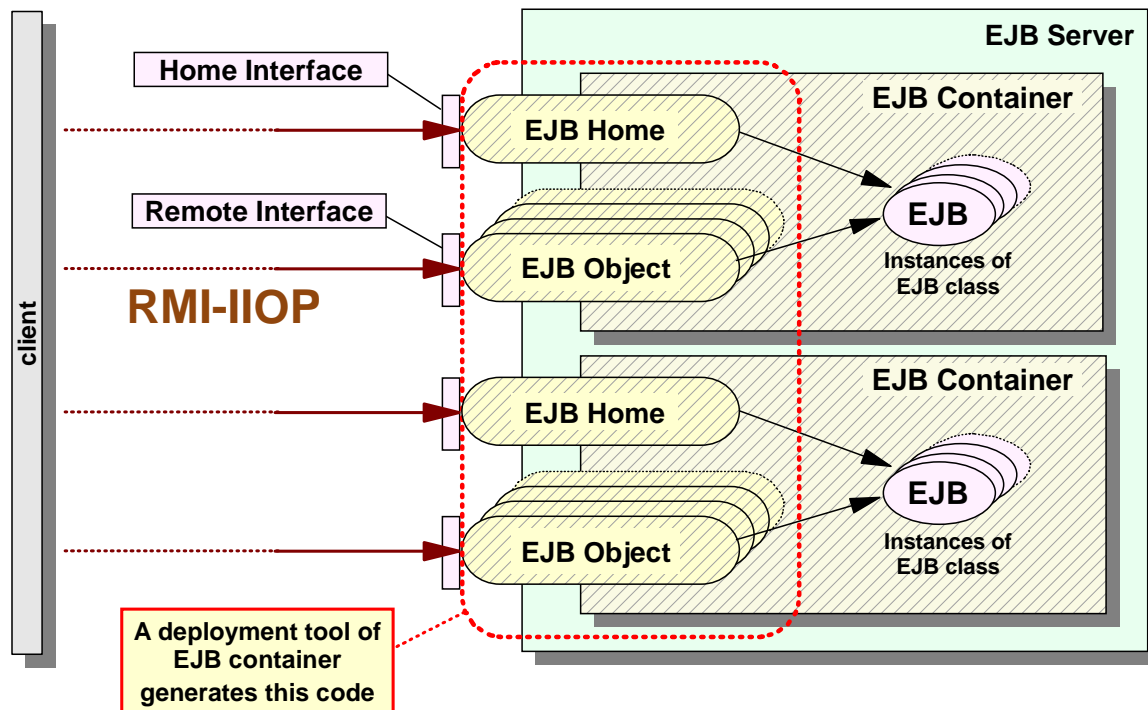
- **CMP (Container-Managed Persistence)** entity bean

- EJB developer need not develop persistence layer
  - An EJB container, such as WebSphere, provides persistence

*Figure 8-3. EJB - Enterprise JavaBeans (1)*

The main two EJBs are the Session and the Entity beans.

## EJB - Enterprise JavaBeans (2)



VA Java V3 - Enhanced EJB

© 1999 IBM Corporation

99SWB402UW

Figure 8-4. EJB - Enterprise JavaBeans (2)1

The client (application) talks to an EJB through two interfaces:

- Home interface, to find and create an EJB
- Remote interface, to invoke methods (functions) in the EJB

The client does not worry about the persistence. This is handled by the EJB or the container.

## EJB Development Environment



### **VA Java Enterprise Edition V2 & Enterprise Update provide you basic EJB development environment**

- You can develop EJB which work in any EJB containers
- You can generate deployed-code for WebSphere Application Server (WAS) Advanced Edition V2

### **VA Java Enterprise Edition V3 focuses on WebSphere Application Server (WAS) Advanced Edition V3**

- You can use all enhanced features provided by VA Java V3 to develop EJB which work in WAS Advanced Edition V3
- But you can *not* use following features to develop EJB which work in other EJB containers *include WAS V2 and WAS Enterprise Edition V3*
  - EJB inheritance and Association
  - Finder Helper and Mapping
    - Deployment task of CMP entity beans
  - Generation of deployed-code

*Figure 8-5. EJB Development Environment*

VA Java is a prime development environment for EJBs. It was introduced in Version 2 with the Enterprise Update (12/98 and 3/99).

Version 3 has improved facilities and matches the support in WebSphere Version 3.

## New Features



Now, we show you new features of EJB Development Environment provided by VA Java Enterprise Edition V3 in the following order :

- **EJB Inheritance Support** (Session/Entity bean)
- **Association Support** (CMP Entity bean)
- **Enhanced ease of use**
  - Properties Pane
  - Create CMP Field SmartGuide
- **Enhancements in Mapping Approach** (CMP Entity bean)
- **Finder Helper Enhancements** (CMP Entity bean)
- **Access Bean** (Session/Entity bean)

*Figure 8-6. New Features*

Now we will go through the new features for EJB development in Version 3.

## Two Forms of Inheritance



**The new EJB Development Environment supports 2 forms of inheritance**

- In VA Java V2 you can only define standard class inheritance

### **Standard class Inheritance**

- EJB class, remote interface, or/and home interface inherit methods from interface/class that are not themselves EJB interface/class
- This defines no inheritance in the sense of EJB hierarchy

### **EJB Inheritance**

- An EJB inherits the followings from an existing EJB
  - CMP fields
  - Associations
  - Methods
  - Method-level control descriptor attributes
- You can define EJB inheritance only within the same EJB group

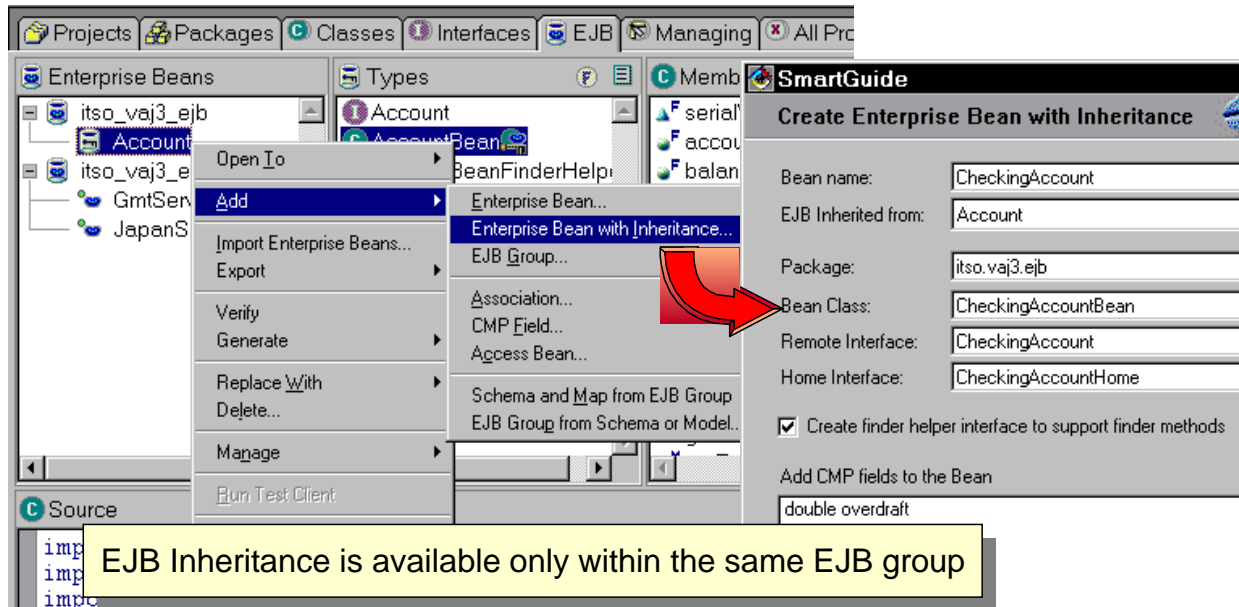
*Figure 8-7. Two Forms of Inheritance*

In Version 3 you can define real EJB inheritance, that is one EJB can inherit from another EJB.

## EJB Inheritance Support (1)



To create a new EJB inherited from other EJB,  
select Add -> Enterprise Bean with Inheritance... from popup



VA Java V3 - Enhanced EJB

© 1999 IBM Corporation

99SWB402UW

Figure 8-8. EJB Inheritance Support (1)

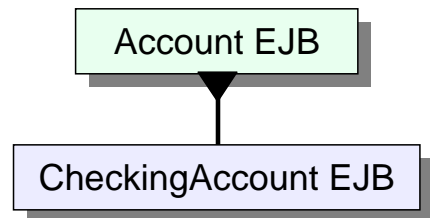
A new menu option is provided to create an EJB with inheritance from another EJB.

## EJB Inheritance Support (2)



Suppose you created **CheckingAccount EJB** inherited from **Account EJB**

- EJB Class (CheckingAccountBean)
  - extends AccountBean class
- Remote Interface (CheckingAccount)
  - extends Account interface
- Home Interface (CheckingAccountHome)
  - does *not* extend AccountHome interface
  - extends javax.ejb.EJBHome
  - Methods defined in AccountHome are migrated to
- Key Class
  - is AccountKey (is *not* CheckingAccountKey)
  - is the same with the key class of Account EJB
  - Key classes are *not* allowed to be inherited from one another



This icon indicate that the EJB inherited from another EJB

Figure 8-9. EJB Inheritance Support (2)

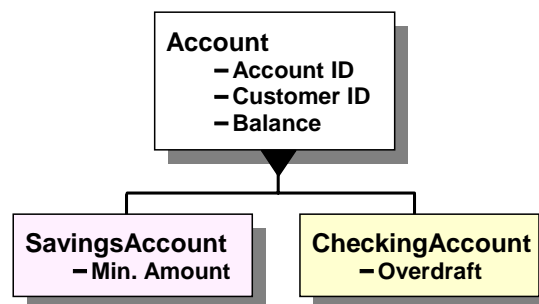
## EJB Inheritance Support (3)



When you map inherited CMP entity beans to database schema using the Map Browser, you can use one of the following mapping styles:

- Single Table Mapping
- Roof-leaf Table Mapping

Suppose you created 2 EJBs inherited from Account EJB, and map them to a database schema



VA Java V3 - Enhanced EJB

© 1999 IBM Corporation

99SWB402UW

Figure 8-10. EJB Inheritance Support (3)

The most important question for a container-managed bean is the mapping to the relational database.

Should you have one single table for all the classes (superclass and subclasses), or should you have a table for each class, with referential constraints between the tables.



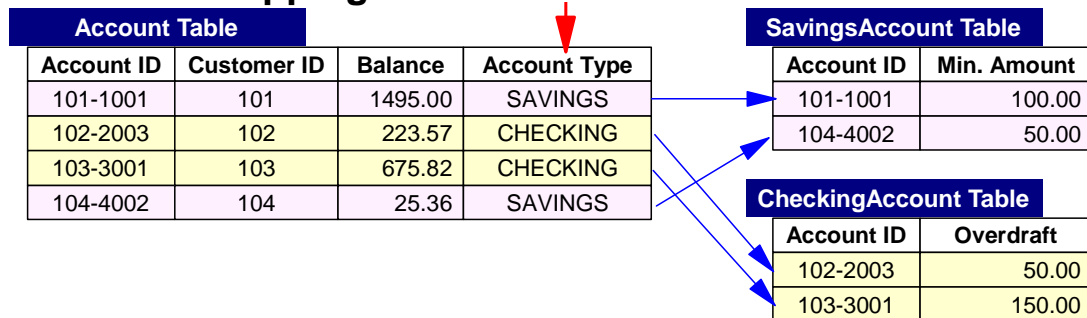
## EJB Inheritance Support (4)



### Single Table Mapping

Account Table					
Account ID	Customer ID	Balance	Account Type	Min. Amount	Overdraft
101-1001	101	1495.00	SAVINGS	100.00	<null>
102-2003	102	223.57	CHECKING	<null>	50.00
103-3001	103	675.82	CHECKING	<null>	150.00
104-4002	104	25.36	SAVINGS	50.00	<null>

### Roof-leaf Table Mapping



VA Java V3 - Enhanced EJB

© 1999 IBM Corporation

99SWB402UW

Figure 8-11. EJB Inheritance Support (3)

Here we show the inheritance mappings.

In the single table mapping we have rows of each class type, with some columns empty.

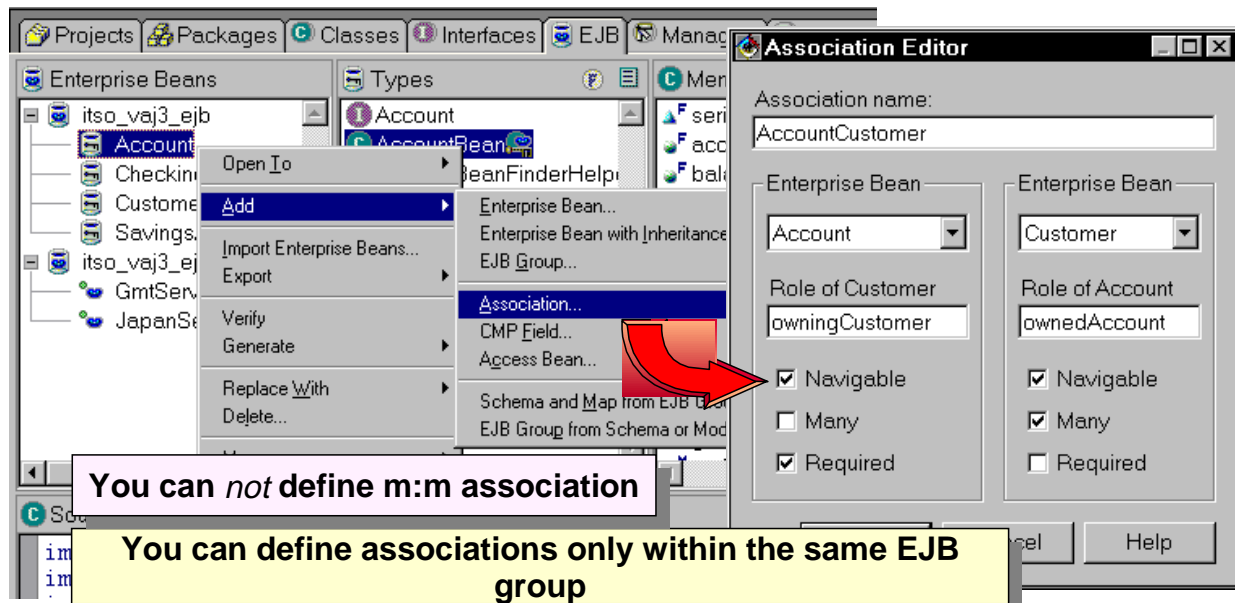
In the multiple table mapping we have a main table with the shared data from the superclass, and a table for each subclass.

Note that in both approaches there is a column that defines the class of the row. This is called the discriminator column.

## Association Support (1)



**You can define an association between CMP entity beans using the Association Editor**



VA Java V3 - Enhanced EJB

© 1999 IBM Corporation

99SWB402UW

Figure 8-12. Association Support (1)

With the association support we define relationships between EJBs. Those relationships will be managed by the system.

The specification includes cardinality, navigable (can you follow), and required.

m:m relationships are not supported. You must define an intermediate EJB and two 1:m relationships.

## Association Support (2)



To support associations, the Association Editor generates the appropriate home and remote methods

**Suppose you created a 1:m association between Account EJB and Customer EJB**

- Account EJB
    - Home methods
      - Enumeration findOwnedAccountsByOwningCustomer( CustomerKey key )
    - Remote methods
      - Customer getOwningCustomer()
      - CustomerKey getOwningCustomerKey()
      - void setOwningCustomer( Customer cust )
      - void setOwningCustomerKey( CustomerKey key )
  - Customer EJB
    - Home methods
      - Customer findOwningCustomerByOwnedAccount( AccountKey key )
    - Remote methods
      - Enumeration getOwnedAccounts()
- 

Figure 8-13. Association Support (2)

## Association Support (3)

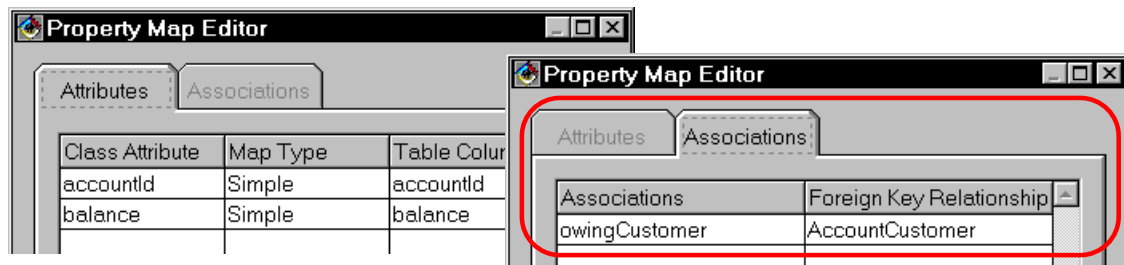


When you map CMP entity beans to database schema, you will map associations to foreign keys

Account Table				Customer Table		
Account ID	Customer ID	Balance	....	Customer ID	Name	Address
101-1001	101	1495.00	....	101	W. Wahli	US
102-2003	102	25.36	....	102	M. Yoshino	Japan
103-3001	103	675	....	103	C. S-Hansen	Denmark
104-4002	104	25.36	....	104	E. Martinez	US

Foreign Key

- You can map associations to foreign keys in the same way you map CMP fields to table columns using the Map Browser



VA Java V3 - Enhanced EJB

© 1999 IBM Corporation

99SWB402UW

Figure 8-14. Association Support (3)

The mapping defines how the relationship is implemented in the tables.

1:m relationship are implemented using a foreign key in one table.

## Properties Pane



### Fields Pane is enhanced and renamed to Properties Pane

- Properties = CMP Fields + Associations + Non-CMP public fields

#### In Properties Pane:

- You can see the following information about CMP entity beans
- You can manipulate/edit them using the popup menu

##### –Key Fields



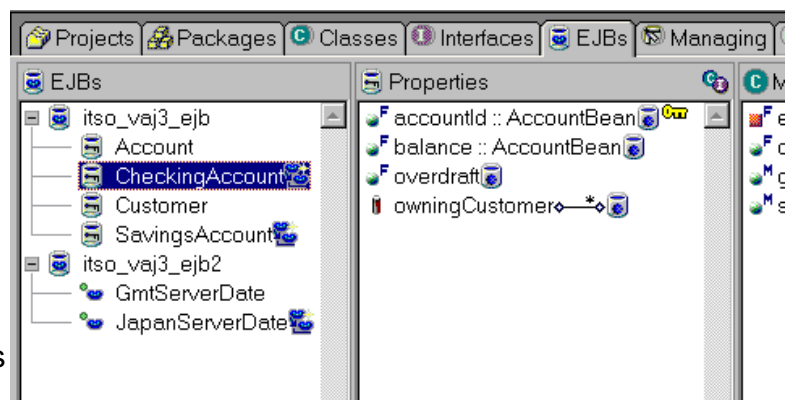
##### –CMP Fields



##### –Associations



##### –Non-CMP public fields [no icon]



VA Java V3 - Enhanced EJB

© 1999 IBM Corporation

99SWB402UW

Figure 8-15. Properties Pane

The properties pane (renamed from fields) shows with icons the kind of property and relationships.

## Create CMP Field SmartGuide



### You can invoke this SmartGuide

- from a popup menu of CMP entity beans
- from Create Enterprise Bean SmartGuide

### By using this SmartGuide

- You can define:
  - Key fields
  - CMP fields
- You can specify:
  - generating accessor methods for a CMP field
  - promoting accessor methods to remote interface

VA Java V3 - Enhanced EJB

© 1999 IBM Corporation

99SWB402UW

Figure 8-16. Create CMP Field SmartGuide

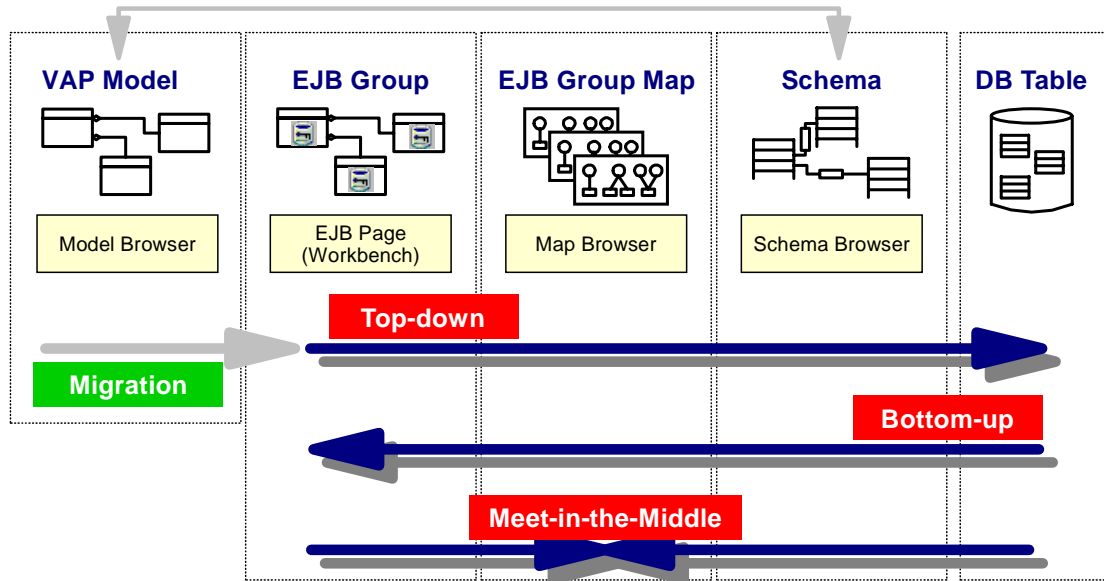
The Create Field SmartGuide makes it easy to define new fields for an EJB.

You can set all appropriate flags for generation of getter/setter methods and adding the methods to the remote interface.

## Enhancements in Mapping Approach



New EJB Development Environment supports the following 3 types of mapping approaches to map CMP entity beans



VA Java V3 - Enhanced EJB

© 1999 IBM Corporation

99SWB402UW

Figure 8-17. Enhancements in Mapping Approach

The mapping is done using the Persistence Builder.

You can develop top-down, starting with the EJB, or you can develop bottom-up starting with a table. Or you can map between an EJB and an existing table.

You can also migrate an existing object model from the Persistence Builder into EJBs.

## Top-down Approach

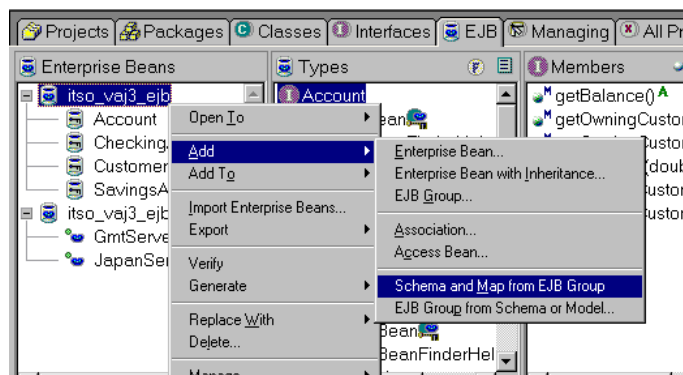


### Assumption

- You have already created CMP entity beans in your EJB group
- You have no database tables to which CMP entity beans are mapped

### Approach

- Create the default map and schema
  - Select **Add -> Schema and Map for EJB Group** from popup menu
- Customize the map and schema using the Map Browser and the Schema Browser
  - EJB inheritance is mapped to single table mapping
- Create database tables using the Schema Browser
- Generate deployed code



VA Java V3 - Enhanced EJB

© 1999 IBM Corporation

99SWB402UW

Figure 8-18. Top-down Approach



## Bottom-up Approach

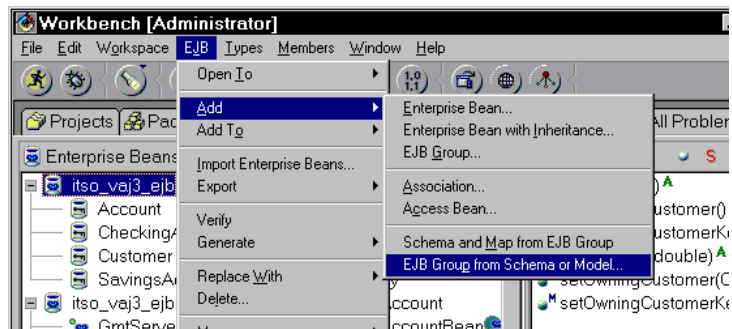


### Assumption

- You already have database tables
- You have no EJB group and CMP entity bean

### Approach

- Import database table definitions to schema using the Schema Browser
- Create the default map, EJB group and CMP entity bean
  - Select **EJBs -> Add -> EJB Group from Schema or Model...** from Workbench menu
- Customize the map and CMP entity beans using the Map Browser and EJB page of Workbench
- Generate deployed code



VA Java V3 - Enhanced EJB

© 1999 IBM Corporation

99SWB402UW

Figure 8-19. Bottom-up Approach

Bottom-up is new in Version 3.

Starting from a tables in a database you can create EJBs that map to those tables, including the relationships between the tables.

## Meet-in-the-Middle Approach



**You can use this approach in VA Java V2**

### Assumption

- You have already created CMP entity beans in your EJB group
- You have already created schema or already have database tables to which CMP entity beans are mapped

### Approach

- When you map your EJB group to existing database tables, import database table definitions to schema using Schema Browser
- Create a EJB group map using Map Browser, and map all CMP fields to database columns
- Generate deployed code

*Figure 8-20. Meet-in-the-Middle Approach*

## Create Default EJB Group SmartGuide



**Create Default EJB Group**  
SmartGuide provides the way to create a new EJB group from:

- VAP (VisualAge Persistence Builder) model
- Schema

**Using this SmartGuide, you can easily convert your VAP model to EJB model**

**To invoke this SmartGuide:**

- Select **EJB -> Add -> EJB Group from Schema or Model...** from menu

SmartGuide

Create Default EJB Group

Project: ITSO VA Java V3 Workshop Browse...

Package: itso.vaj3.ejb.vap Browse...

Create a new EJB group named:  
itso\_vaj3\_ejb\_vap

☐ Create from schema

Available Schemas:

VA Java Enterprise Edition V3 provides you convergence between VisualAge Persistence and EJB

☒ Create from model

Available Persistence Builder Models:  
VAPBankModel

Figure 8-21. Create Default EJB Group SmartGuide

This new Version 3 function allows to convert a Persistence Builder model into an EJB group.

## Finder Helper Enhancement



To implement a custom finder for CMP entity bean, you can use one of the following 3 forms (2 forms)

### SELECT Custom Finder

- Define a whole SELECT statement in the Finder Helper interface

```
public static final String <method-name>QueryString
    = "SELECT ... FROM ... WHERE T1.BALANCE > ?";
```
- Actually, you can not customize "SELECT .. FROM .." part
- *Do not use this form for new development*

### WHERE Custom Finder

- Define a WHERE clause in the Finder Helper interface
- This is the replacement of SELECT custom finder

### Method Custom Finder

- Define a method signature in the Finder Helper interface
- This form is the most flexible, but you have to write Java code

Figure 8-22. Finder Helper Enhancement

These enhancements make it easier to create custom SQL access methods for an EJB.

## Implement WHERE Custom Finder



**What you have to do is define a WHERE clause in the Finder Helper interface**

- `public static final String <method-name>WhereClause = "T1.BALANCE > ?";`
- Use "T1", "T2", ... as aliases for tables

**You are free from maintaining "SELECT .. FROM ..." part**

- If you use SELECT custom finder, you have to maintain this part
- This part is affected of the mapping changes directly

**Limitations:**

- You have to maintain the column names in WHERE clause
- The order and number of find() method parameters have to be the same with the order and number of "?" in the WHERE clause
- find() method parameters have to be base type or base class
  - You can not use your custom classes as find() method parameters

*Figure 8-23. Implement WHERE Custom Finder*

Define the WHERE clause in AccountBeanFinderHelper class:

```
final static String findBalanceGreaterThanWhereClause = "T1.BALANCE > ?";
```

Define method signature in AccountHome interface:

```
java.util.Enumeration findBalanceGreaterThan(double balance)  
    throws java.rmi.RemoteException, javax.ejb.FinderException;
```

In client code use the narrow method to access the result EJB:

```
java.util.Enumeration enum = AccountHome.findBalanceGreaterThan(100.00);  
while (enum.hasMoreElements()) {  
    Object o = enum.nextElement();  
    Account a = (Account)javax.rmi.PortableRemoteObject.narrow((org.omg.CORBA.Object)o,Account.class);  
    ....  
}
```

## Implement Method Custom Finder (1)



### What you have to do are:

- Define a method signature in the Finder Helper interface
  - Return type is `java.sql.PreparedStatement`
  - Name and parameters are the same with `find()` method
    - `public java.sql.PreparedStatement`  
`<method-name>( <p1>, <p2>, ... ) throws Exception;`
- Create a custom finder class:
  - which extends `VapEJSJDBCFindObject` and implements the Finder Helper interface
  - in the same package as the deployed code
  - whose name is **<ejb-name>BeanFinderObject** or any name
    - If the name is not `<ejb-name>BeanFinderObject`, you have to specify the class name using **CustomFinderClassName** environment properties
- Implement the method
  - To create a prepared statement, use the following methods provided by the super class (`VapEJSJDBCFindObject`)
    - `getMergedPreparedStatement()`, `getMergedWhereCount()`
    - `getPreparedStatement()`, `getGenericFindSqlString()`, `getGenericFindInsertPoints()`

Figure 8-24. Implement Method Custom Finder (1)

## Implement Method Custom Finder (2)



### Sample:

- This sample works as the same as WHERE Custom Finder defined:

-findBalanceGreaterThanWhereClause = "T1.BALANCE > ?"

```
public class <ejb-name>BeanFinderObject
    extends com.ibm.vap.finders.VapEJSJDBCFinderObject
    implements DepartmentBeanFinderHelper {

    public PreparedStatement findBalanceGreaterThan( double value )
        throws Exception {
        String whereClause = "T1.BALANCE > ?";
        int paramCountInWhereClause = 1;
        int mergeCount = getMergedWhereCount();
        PreparedStatement ps = getMergedPreparedStatement( whereClause );

        for ( int i = 0; i < mergeCount * paramCountInWhereClause;
              i += paramCountInWhereClause ) {
            ps.setDouble( i + 1, value );
        }
        return ps;
    }
}
```

Figure 8-25. Implement Method Custom Finder (2)

## Access Bean



### To access EJB, you have to:

- Obtain a context to the name server
- Look up the home of EJB
- Create/Find an EJB instance

### VA Java V3 provides you an easy way to access EJB by generating so called Access Bean

- To access EJB using an Access Bean, you only have to create an instance of it
- Access Beans are exported to the client JAR file and marked as JavaBean

### You can create 3 kind of Access Bean

- **Beanified Wrapper** (Session and Entity)
- **Copy Helper** (Entity only)
- **Rowset** (Entity only)

VA Java V3 - Enhanced EJB

© 1999 IBM Corporation

99SWB402UW

*Figure 8-26. Access Bean*

In Version 2 accessing an EJB took some code to get the context of the name server, look up the home interface of the EJB, before you could find or create an EJB instance.

In Version 3 VA Java provides access beans that take over some of the work.

Three kind of access beans provide different levels of support.



## Beanified Wrapper



- **Simply wraps home interface and remote interface of EJB**
  - All methods call to this bean are delegated to EJB home or object
  - It is serializable, but reference to EJB object is transient
- **Has all methods defined in remote interface**
  - If *return type* is a remote interface of EJB, the *return type* will be changed to the type of corresponding Access Bean
- **Has constructors to which create/find methods defined in home interface are mapped**
  - You can select which create/find methods should be mapped to the default (no argument) constructor of Access Bean
  - Parameters for create/find method which is mapped to a default constructor are mapped to **init\_argXxx** properties
- **Has find methods defined in home interface which return a collection of EJB Objects**

VA Java V3 - Enhanced EJB

© 1999 IBM Corporation

99SWB402UW

Figure 8-27. Beanified Wrapper

The simplest access bean is the Beanified Wrapper.

Calls to this wrapper are delegated to the EJB home or object.

You can map either a find or a create method to the constructor. The EJB is not instantiated at constructor time. After the constructor has run, you pass a key to the bean using the `init_argXxx` method to set a key value. At first access (get/set) the EJB is instantiated (lazy instantiation).

## Copy Helper



**This bean has all characteristics of Beanified Wrapper**

**This bean caches property values of EJB**

- Access to property values become first, because it is local call
  - getXxx/setXxx methods do *not* throw Remote Exception, if its value is already cached
- All selected property values are cached:
  - when you call one of getXxx methods at the first time
  - when you call refreshCopyHelper method of this bean
- Cached property values are written back to EJB object:
  - when you call commitCopyHelper method of this bean
- You can select which properties should be cached

**To support Copy Helper, remote interface and EJB class are automatically updated by VA Java**

- Remote interface extends com.ibm.ivj.ejb.runtime.CopyHelper interface
- EJB class implements all methods defined in CopyHelper interface

*Figure 8-28. Copy Helper*

The Copy Helper does more work the beanified wrapper.

The copy helper caches all properties at first access or when refresh is called. Updated properties are propagated on commit.

## Rowset



### **This bean holds a collection of Copy Helper instances**

- If you create Rowset, Copy Helper also is created
- You can not invoke any methods of home/remote interface through this bean

### **What this bean provides you are functions to maintain a collection of Copy Helper instances**

*Figure 8-29. Rowset*

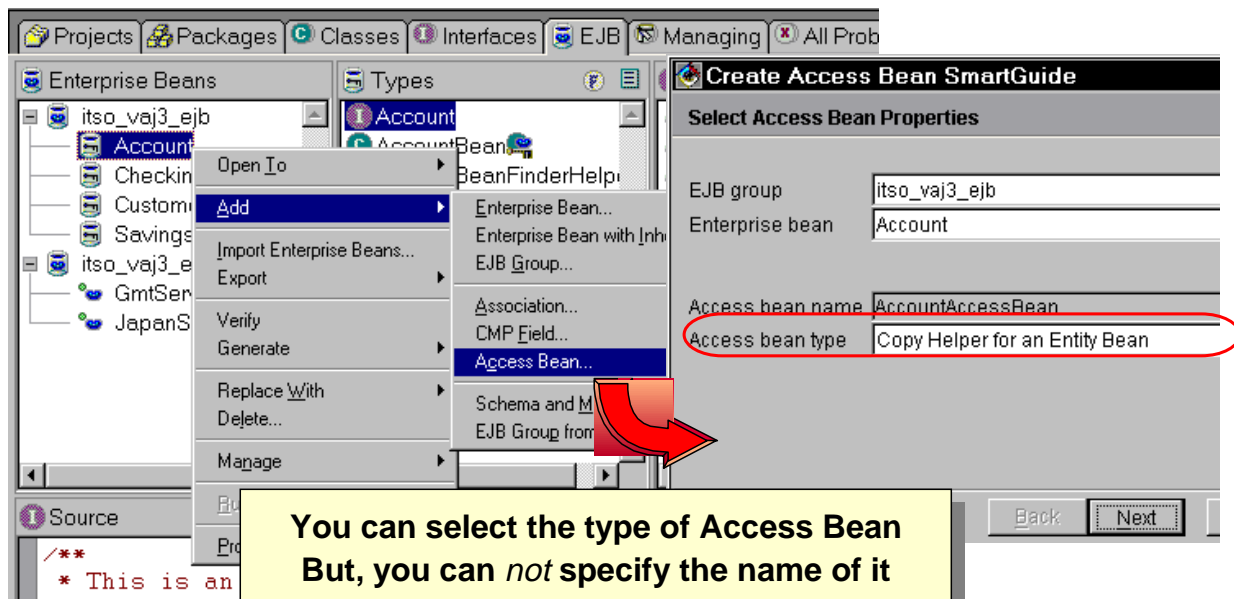
The Rowset holds a collection of copy helpers.

You can retrieve a set of copy helpers with one call!

## Generating Access Bean (1)



To create Access Bean, select Add -> Access Bean... from popup menu



VA Java V3 - Enhanced EJB

© 1999 IBM Corporation

99SWB402UW

Figure 8-30. Generating Access Bean (1)

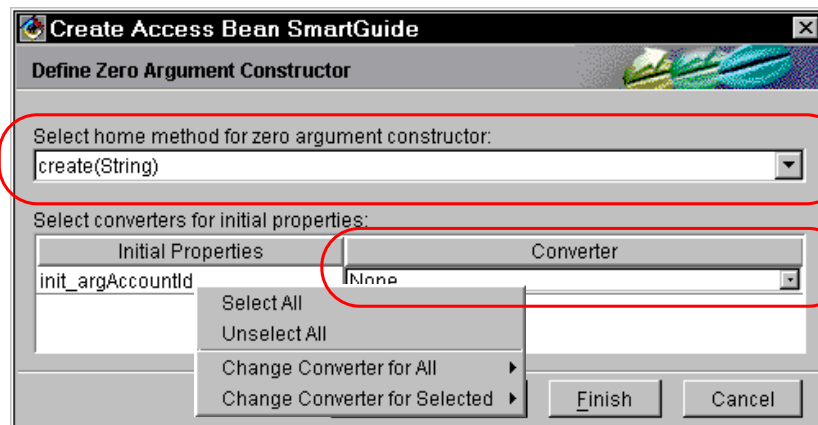
A SmartGuide is provided to create the access beans. You specify the type of bean in the SmartGuide.

## Generating Access Bean (2)



**You can specify which create/find method should be mapped to the default (no argument) constructor**

- Parameters of create/find method which is mapped to a default constructor are mapped to init\_argXxx properties
- You can also select a converter for these properties



*Figure 8-31. Generating Access Bean (2)*

You have to decide what the default constructor does: either find or create.

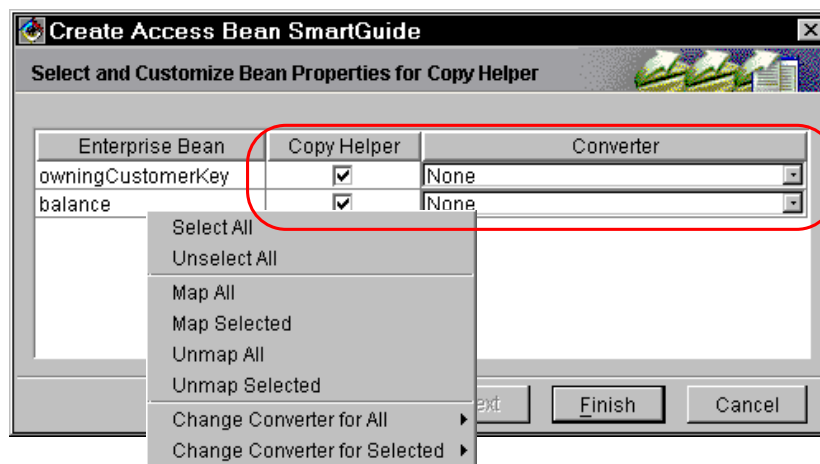
## Generating Access Bean (3)



**When creating Access Bean, you can specify:**

- Which properties of entity bean should be cached by Copy Helper
- Converter

**To manipulate many properties, you can use popup menu**



VA Java V3 - Enhanced EJB

© 1999 IBM Corporation

99SWB402UW

*Figure 8-32. Generating Access Bean (3)*

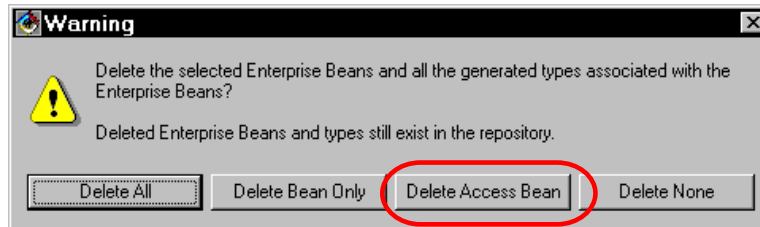
For the copy helper you can choose which properties should be cached. Others would be held in the EJB only and calls be forwarded.

## Deleting Access Bean



To delete Access Bean,

- select **Delete...** from popup menu of EJB Group or EJB
- and select **Delete Access Bean** button from the following dialog



- Don't delete Access Bean by deleting its class from Workspace
  - This operation causes errors in EJB

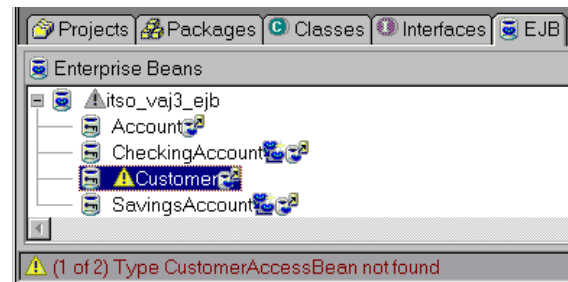


Figure 8-33. Deleting Access Bean

To delete access beans you must use the special facility and not delete from the workspace. This is important because other code must be updated.

## Converter



**Some EJB clients, such as JSP, prefer to access EJB's properties as `java.lang.String`**

**By specifying converters when you create Access Bean, you can access EJB's properties as `java.lang.String`**

**VA Java provides `SimpleStringConverter`**

- This converter can convert all basic type/class to `java.lang.String`

**You can create a custom converter**

- Create your custom converter class that implements `com.ibm.ivj.ejb.runtime.StringConverter` interface
- Define the following methods in your custom converter class
  - `public static String <type>ToString( <type> value )`
  - `public static <type> StringTo<type>( String value )`
- Custom converters are automatically shown in the converter list

*Figure 8-34. Converter*

Using converters you can map EJB attributes to data types that are simpler to handle, for example, in JSPs.



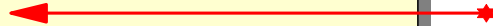
## Instantiating Access Bean (1)



### By using default constructor

- You have to set parameter values for create/find methods using setInit\_argXxx methods
- EJB object is created/found when you call a method mapped to remote method

```
CustomerAccessBean cust = new CustomerAccessBean();  
cust.setInit_argCustomerId( "123" );  
String name = cust.getName();
```



### By using constructors which have arguments

- EJB object is created/found when a instance of Access Bean is created.

```
CustomerKey key = new CustomerKey( "123" );  
CustomerAccessBean cust = new CustomerAccessBean( key );  
String name = cust.getName();
```



Figure 8-35. Instantiating Access Bean (1)

When instantiating an access bean, remember that the default constructor either calls the find or the create method, later, when accessing the first attribute, and after the key has been set.

Constructors with parameters are for create only.

## Instantiating Access Bean (2)



**You can specify the URL and type of Name Service using:**

- URL:     `setInit_NameServiceTypeName( String )`
- Type:     `setInt_NameServiceURLName( String )`

**If you want, you can also specify the JNDI name of EJB home using:**

- `setInit_JNDIName( String )`

### **Note:**

- Access Bean uses these properties to create/find EJB object, so you have to set these properties before you call remote method
  - This means, you can not use constructors that have parameters, if you want to specify these properties
- WebSphere run time provides a run-time API to automatically create the default rootContext. In this case, you need not specify the URL and type of Name Service

*Figure 8-36. Instantiating Access Bean (2)*

Special parameters allow you to tailor how the EJB is located.

## Advantage of Access Bean



### Advantage

- You are free from tedious part of EJB programming
- You can access all EJB properties as `java.lang.String`
- Copy Helper
  - You can improve performance
    - You need not to write additional code to improve property accessing performance
  - No `RemoteException` is thrown while accessing cached properties

### Caution

- Copy Helper
  - Even if you access only one property, all selected properties are retrieved from EJB object and they are written back to EJB object
  - You have to call `commitCopyHelper` to commit cached properties

### Disadvantage

- EJB inheritance is not reflected to Access Bean
  - `CheckingAccountAccessBean` is not a subclass of `AccountAccessBean`

VA Java V3 - Enhanced EJB

© 1999 IBM Corporation

99SWB402UW

*Figure 8-37. Advantage of Access Bean*

Access beans make EJB programming simpler.

For example, you can use an access bean in a JSP. The access bean is instantiated automatically. Then you pass the key (the `init_xxxx` method), and you retrieve attributes. Everything happens automatically behind the scene.

## Starting EJB Server (1)

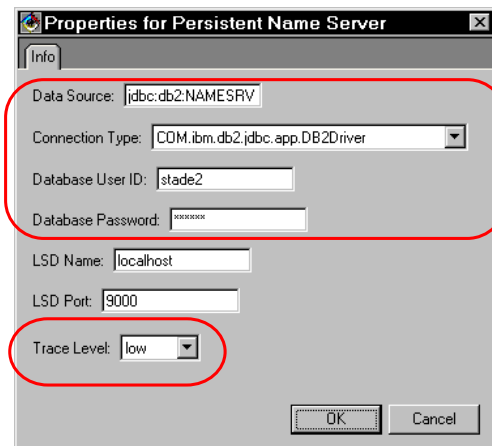


**New Persistence Name Server (PNS) stores information to database**

**When you start PNS at the first time, you have to :**

- Create a database
  - You need not create any tables
- In property window of PNS, specify :
  - JDBC URL
  - JDBC Driver
  - Database user ID and password
- Can use **InstantDB** instead of real database

**In VA Java V3, you can specify the trace level of PNS**



*Figure 8-38. Starting EJB Server (1)*

Starting the EJB server is several steps.

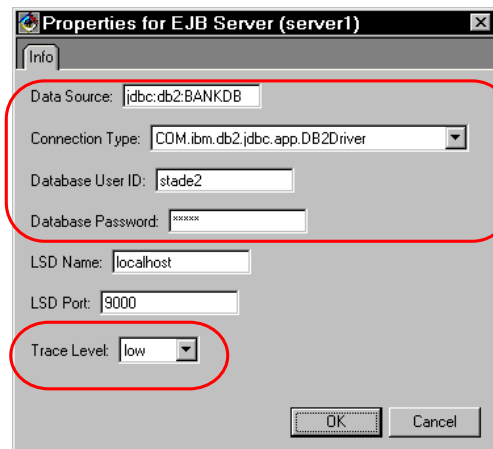
First the name server must be configured (once) and started.

## Starting EJB Server (2)



**If a EJB Group contains CMP Entity Beans, you must finish the following tasks before you start EJB Server at the first time**

- Create a database
  - If a database does not exist, you have to create it by yourself
- Create tables
  - If tables does not exist, you have to create them using the Schema Browser
- In property window of the EJB Server, specify :
  - JDBC URL
  - JDBC Driver
  - Database user ID and password
  - In VA Java V3, you can specify the trace level of EJB Server



*Figure 8-39. Starting EJB Server (2)*

The EJB server must be configured as well.

A database must be specified. The tables in the database (for the EJBs) must exist or they can be created using the Persistence Builder Schema Browser.

## Exporting EJB to JAR file



You can export your EJBs to JAR file in the same way with V2

- **EJB JAR**

- contains no deployed code
- you can use this option to create EJB JAR file which is deployed to any EJB Server include WAS AE V3 and WAS EE V3

- **Deployed JAR**

- contains deployed code for WAS AE V3
- if you mapped your CMP Entity Beans, you must use this option to prevent WAS from recreating the deployed code
- EJBs that exist in an inheritance or association must be packaged in the same JAR file

- **EJB JAR for CB**

- contains no deployed code
- this option is available when you selected one EJB Group
- if you will deploy your EJBs to WAS EE V3 and you have already installed Component Broker (CB) tools in your machine, you should use this option

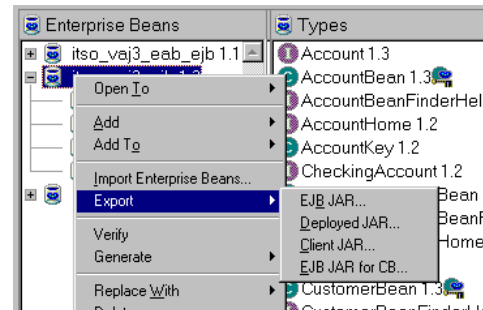


Figure 8-40. Exporting EJB to JAR file

Deployment of the EJBs has not changed from Version 2.

You either export as EJB Jar or Deployed Jar.

- ☐ With the EJB Jar, WebSphere will do most of the deployment work. A new mapping is created at this time.
- ☐ With the Deployed Jar all the mapping is retained from the VA Java environment.
- ☐ A third option is for deployment into Component Broker.

## Summary



### **VA Java V2 & Enterprise Update provide you :**

- basic features to develop and test EJB
- no feature to develop EJB clients

### **VA Java V3 provides you :**

- more powerful features
  - EJB Inheritance, Association
  - Enhancements in FinderHelper, Mapping
- a easy way to develop EJB clients
  - Access Bean

**Using VA Java V3, you can easily develop EJB that run in WebSphere Application Server Advanced Edition V3**

*Figure 8-41. Summary*

The EJB development environment provided by VisualAge for Java has improved with Version 3.

With the integration with WebSphere Version 3 a straight-forward path from development to execution is implemented.

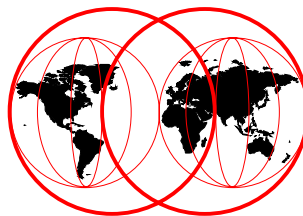




# **9 VA Java Version 3 Enhanced Compiler & Distributed Debugger**

**VisualAge for Java Version 3**

Enhanced HPJ / Distributed Debugger



© 1999 IBM Corporation

99SWB402UW

## Objectives



### **What is the concept behind High-Performance Java Compiler?**

- HPJ: an Overview
- Differences to VA Java 2.0

### **The new debugger of VA Java 3.0**

- Distributed debugger
  - Installation
  - Concepts
  - Local and remote debugging
- ObjectLevelTrace

*Figure 9-2. Objectives*

## HPJ Overview



### HPJ is a part of the ET/Workstation of VA Java

- Use HPJ for creating applications running on S/390, AIX, NT, OS/2
- HPJ compiles byte code or Java source code into optimized, platform specific native code (executables or DLLs)
- Programs compiled with HPJ are faster than with a JIT
- HPJ supports JNI
- HPJ is available inside VA Java IDE, and from the command line

### HPJ Command: **hpj [options] input-files**

```
hpj -exe|-jll|-jlc -O -follow -g -check none -o name.exe name.java
```

- Execution from EXE  
set IBMHPJ\_OPTS=options  
name.exe
- Execution from DLL  
hpjava -load d:\BMV\Java\hpj\lib\swingall.jll package.classname

*Figure 9-3. HPJ Overview*

The High-Performance Java Compiler is shipped with the ET/Workstation feature.

It is functionally the same as in Version 2, with improved performance.

Java classes can be compiled from inside the VA Java IDE, or in the file system from the command line.

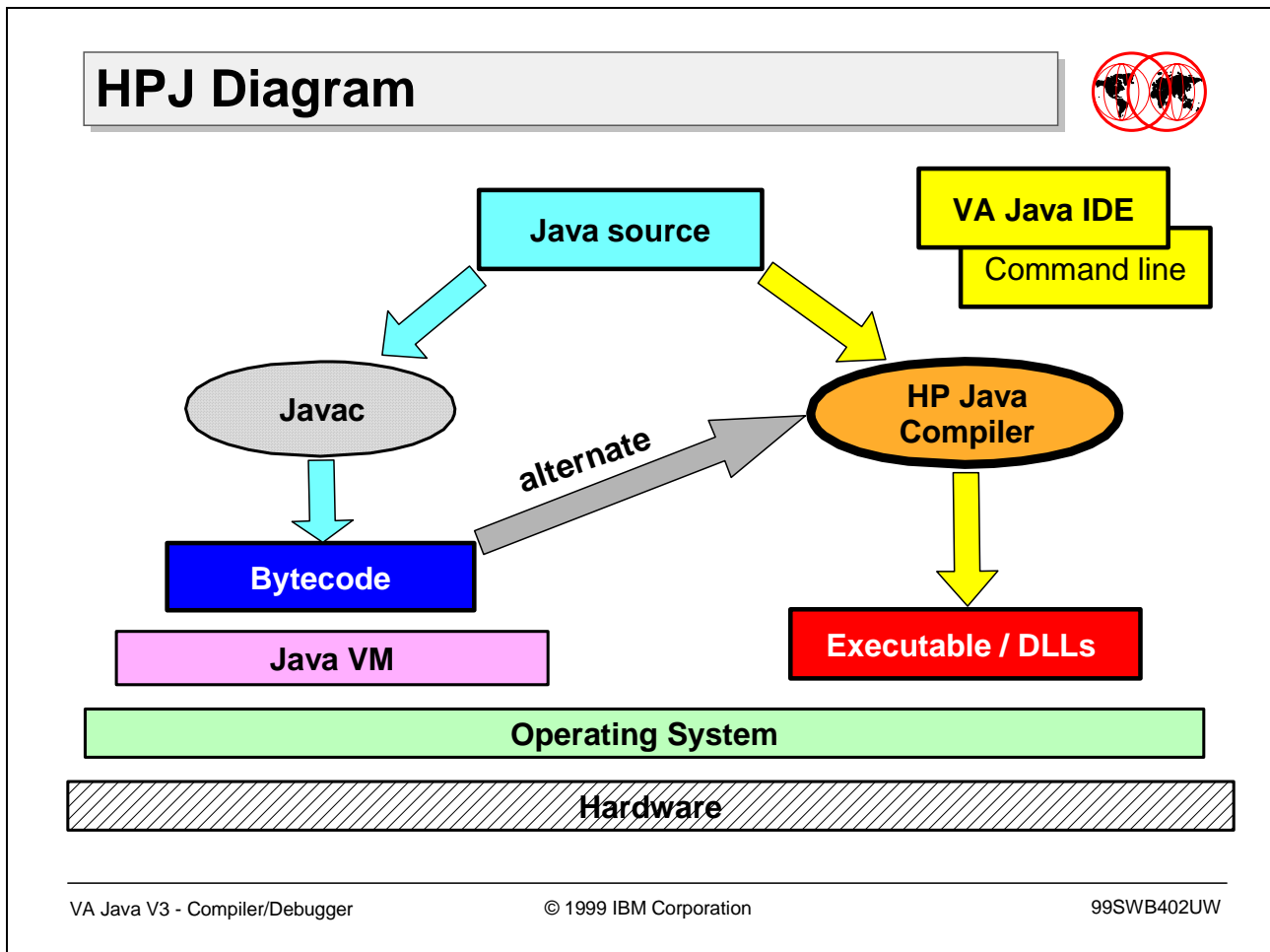


Figure 9-4. HPJ Diagram

On the left in the diagram you see the normal way Java source code (.java) is compiled into byte code (.class). That code is interpreted by a Java Virtual Machine (JVM) at runtime.

On the right side is the HPJ compiler. It compiles either source code or byte code into an executable or a DLL. The resulting code is platform specific.

## Differences in HPJ: VA Java V2 to V3



**There are no differences regarding the invocation of HPJ from VA Java 2.0 compared to VA Java 3.0**

- all command line options are the same
- the settings inside the IDE are still the same

### **New in VA Java 3.0 HPJ:**

- an error message is issued , if an error occurs while comp. time
- the message format: IVJHnnnn(L) text <&n>
  - L - is one of four error severity levels:
    - I - informational message
    - W - warning message
    - E - error message
    - S - severe error message
  - text - the text of the error message
  - <&n> - a variable substituted with an appropriate value, such as a class name or file name
- a message reference is available via online help system

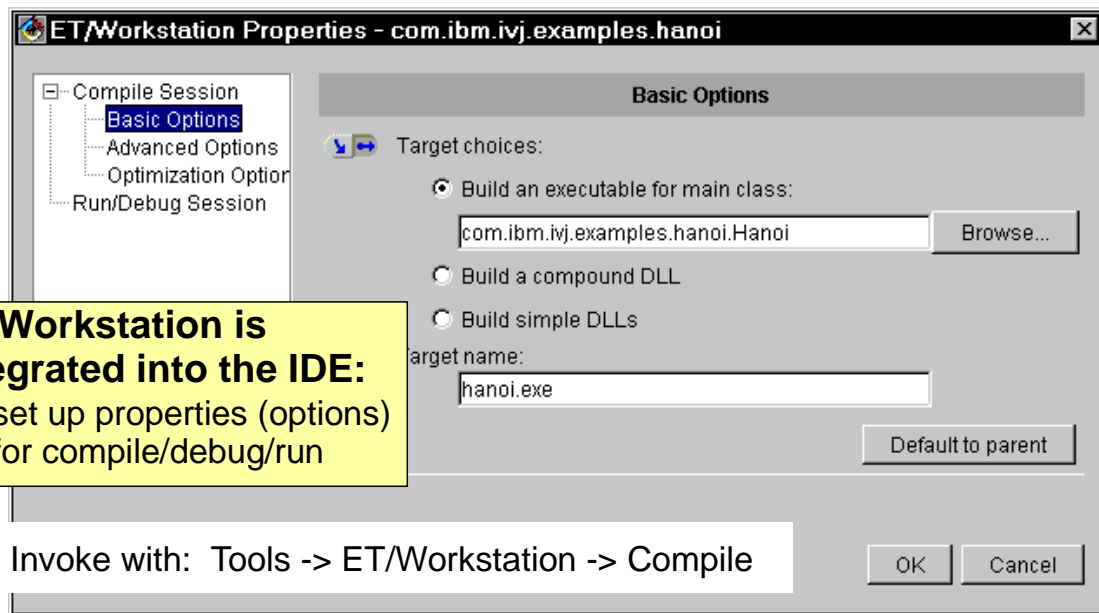
*Figure 9-5. Differences in HPJ: VA Java V2 to V3*

## Calling HPJ within VA Java



### Must setup properties before invoking HPJ

- Tools -> ET/Workstation -> Properties (for class, package, project)



- Invoke with: Tools -> ET/Workstation -> Compile

VA Java V3 - Compiler/Debugger

© 1999 IBM Corporation

99SWB402UW

Figure 9-6. Calling HPJ within VA Java

Before compiling a class inside the IDE, you set up the properties.

For a single class without references in the same package, you can set the properties for the class itself.

If multiple classes in the package have to be compiled together, you set the properties for the package and you name the main classes that is executed.

On subsequent pages you can set all the other compile options.

You can also set properties for the debugger and runtime.

## Distributed Debugger Overview



### The IBM Distributed Debugger is a client/server application

- Debug local applications as well as applications accessible via a network
- Local debugging
  - debug server (the debug engine) and the program to debug, runs on your local machine
- Remote debugging
  - debug engine and the program to debug, runs on a system in the network
- The debug engine can be attached to running programs or can start programs itself
- The debug client is a GUI for executing commands against the debug engine
  - set breakpoints, see variable values, step through code
  - the debug client allows to debug several programs at the same time
  - each debugged program is shown on a separate page

**Deployment Debugger - multi-language/multi-platform**

*Figure 9-7. Distributed Debugger Overview*

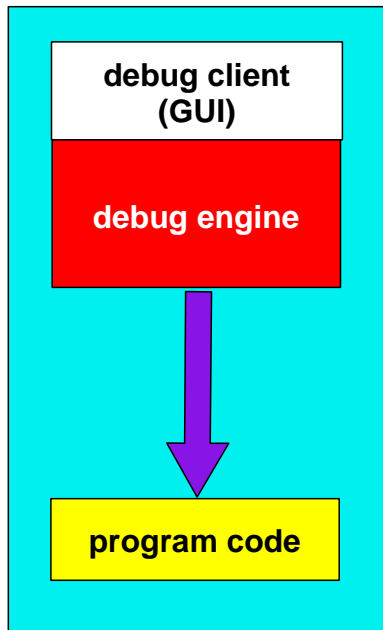
The Distributed Debugger is a new component. It is separately installed.

You can debug local or remote applications. The GUI is on the local system and it controls the execution of the program running on the local or remote system.

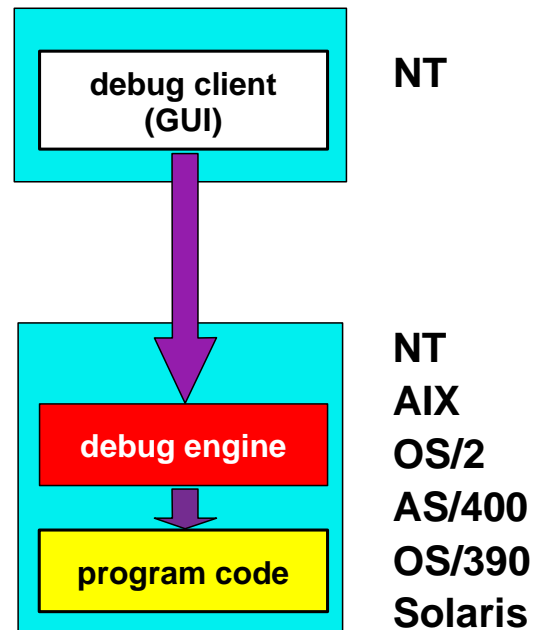
## Debugging Schema



### Local Debugging



### Remote Debugging



VA Java V3 - Compiler/Debugger

© 1999 IBM Corporation

99SWB402UW

Figure 9-8. Debugging Schema

The debug client on the local system is the same in both modes of debugging.

The debug engine runs on the system where the target program runs.



## Languages Supported by Debugger



**The distributed debugger supports debugging of compiled platform specific code as well interpreted java code**

- Supported languages for compiled code
  - C
  - C++
  - FORTRAN
  - COBOL
  - PL/I
  - Java (compiled with HPJ)
- Supported interpreted language
  - Java
- All further statements will concentrate on debugging Java code

*Figure 9-9. Languages Supported by Debugger*

The distributed debugger supports a number of programming languages.

Note that it supports both interpreted and compiled Java code.

## Installing the Distributed Debugger



### The Distributed Debugger has two separate install routines

- You can use the installation routine within the VA JAVA setup screen
  - start d:\vajavav3\setup.exe
  - choose Install Distributed Debugger from the menu
- Or go to the directory d:\vajavav3\extras\Debugger\y
  - d is your CD ROM drive
  - y is the OS, where the program runs to debug
    - y may be one choice of AIX, OS/2, OS390, Solaris ,or Windows
  - see the readme files for the platform specific install steps
    - on windows, simply starts the setup.exe in d:\vajavav3\extras\Debugger\windows

### The debugger is started outside of VA Java

- The debugger will be in a directory **c:\IBMDebug**, if no other path was specified during installation
- program icons are created in the VA Java folder
- use the debugger commands **idebug**, ... from a command line window to setup/start debugger engine and client

*Figure 9-10. Installing the Distributed Debugger*

## Setup the Debugging Environment (1)



### The following variables may be set for debugging

- **DER\_DBG\_CASESENSITIVE** workstation environment variable
  - If set to a not NULL value (1, "True", "whatever"...), the debugger compare part names and module name case-sensitive
  - Default value is converting to uppercase
  - This variable does not effect file system access
- **DER\_DBG\_LOCAL\_PATH** environment variable
  - Used to locate executables and DLLs on the system where you are debugging your program
  - Only available on AIX and WIN, for OS/2, use the PATH and LIBPATH variables
- **DER\_DBG\_NUMBEROFELEMENTS** workstation environment variable
  - An integer value, that tells the debugger the max. number of elements to display for an array or structure
- **DER\_DBG\_OVERRIDE** environment variable
  - overrides temporary the DER\_DBG\_LOCAL\_PATH

*Figure 9-11. Setup the Debugging Environment (1)*

## Setup the Debugging Environment (2)



- DER\_DBG\_TAB environment variable
  - Integer value, indicating the number of spaces to convert a tab char. into in a source or mixed view into number of spaces.
- DER\_DBG\_TABGRID environment variable
  - overrides the DER\_DBG\_TAB value
  - Integer value, indicating how the debugger uses tabs in the source to align with columns in the source or mixed view pane
  - For Example, DER\_DBG\_TABGRID is set to 4, tab stops are set at 4,8,12,16....
- PATH system variable
  - used to locate the debugger, the program to debug
  - on WIN also used to locate DLLs
- DPATH system variable
  - used on OS/2 to locate the message files used by the debugger
- CLASSPATH environment variable
  - tells the debugger, where are the classes found to debug

*Figure 9-12. Setup the Debugging Environment (2)*

## Starting the Debug Process



### The Distributed Debugger provides local and remote debugging

- **Local debugging:**
  - **idebug** command from a command line
- **Remote debugging:**
  - Point the CLASSPATH variable on the remote system to all classes which are needed for running your program you want to debug
  - Start the **irmtdbgj** for interpreted java classes, **irmtdbg** for any compiled program on the remote system command line
  - On your local system, start the debug client with the **idebug** command

### Idebug command:

- `idebug [idebug_options] [local_debug_parameters | remote_debug_parameters | ui_daemon_parameters] [--] [program_name [program_parameters]]`
- For a complete list of the parameters, see the online help
- If no parameters are provided, you are prompted

*Figure 9-13. Starting the Debug Process*

On the local system you always start the **idebug** program.

For remote debugging you start the engine on the target system. Depending if you debug interpretative code or compiled code you start the **irmtdbgj** or **irmtdbg** program.

## Debugger Start Dialog



Click here to  
debug Java  
code

Enter the class name  
here or browse for it

Enter here the  
parameters to the class

Enter here Arguments  
passed to the JVM

Specify local or  
remote debugging

Finally, click Load

Load program

Compiled (Intel/AX) | Interpreted (Intel/AX) | AS/400

Dominant language the debugger will use:  
Interpreted Java Advanced...

Enter the class name (without .class) containing main to be invoked:  
com.ibm.lvj.examples.vc.beandemo.BeanDemo Browse...

Enter any program parameters:  
[Empty text box]

Enter the program JVM arguments (if any):  
[Empty text box]

Select startup behavior:  
☒ Use program profile  
☐ Debug program initialization

Select local or remote execution of the program:  
☒ Local  
☐ TCP/IP Connection

Host: localhost Port: 8000

Load Cancel Help

VA Java V3 - Compiler/Debugger

© 1999 IBM Corporation

99SWB402UW

Figure 9-14. Debugger Start Dialog

When the debugger (idebug) is started a dialog prompts you for the necessary parameters.

## Attaching to Running Processes (1)



### Attaching to a local running process

- Only available for native programs or HPJ compiled Java Code
- Use the idebug -a on command line
- Or menu  
File -> AttachProcess  
on the debuggers main window
  - select the process you want to debug from the process list
  - enter the full path name in the ProcessPath field

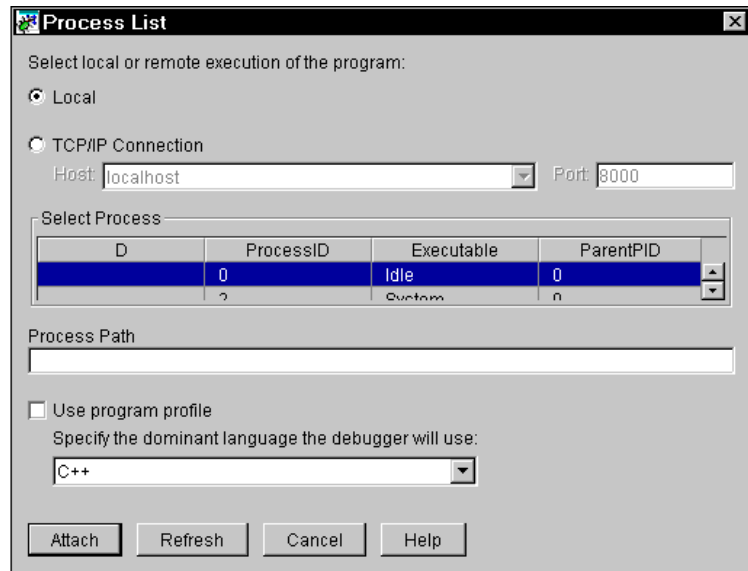


Figure 9-15. Attaching to Running Processes (1)

You can even attach the debugger to a running program and start debugging that code.

## Attaching to Running Processes (2)



### Attach to a remote running Process

- On the command line of the remote machine, run the `irmtdbg` program
  - If you specified a `-qport` option, write down the port number
- On the local machine, you can start the debugger
  - From commandline with the command
    - `idebug -ghost= [-qport=] -a`
  - Or from the menu **File > Attach Process**
    - select TCP/IP Connection
    - Fill in Host and Port (from the `-qport` option)
    - Click on Refresh
    - Select the process to debug from the Process list
    - Enter the full path name into the ProcessPath field

*Figure 9-16. Attaching to Running Processes (2)*



## Attaching to Running Processes (3)



### Attach to a local/remote running JVM

- Attaching to a JVM is only possible for interpreted Java code
  - For HPJ compiled Java code use the AttachProcess function
- You have to run your Java program via java\_g - debug on a the command line.
  - A password will be generated from java\_g, take note of this password
- Attach to the JVM via command line
  - irmtdbgj -qport= -host= -password=
    - -qport and -host are only for remote JVM debugging
  - use the idebug -a option, processID=0 for attaching to a JVM
- Attach to the JVM via GUI
  - Select menu File > Attach JVM
  - Select: Use a remote engine
  - Fill in the EngineHost field the Host Name and to the Port field the portnumber

*Figure 9-17. Attaching to Running Processes (3)*

## Debugging an Applet



### Debugging Applets always starts with debugging the `sun.applet.AppletViewer` class

- start from the command line:  
`idebug -qlang=java sun.applet.AppletViewer`
  - The may be
    - a URL (`HTTP://` or `file://`)
    - a filename , then the `idebug` command have to be issued from the directory, where the applet is located
- Click on OK
- Select Menu Source -> Open New Source
- Enter the name of the applet class you want to debug
- Click on OK

*Figure 9-18. Debugging an Applet*

The debugger can be used to debug an applet running in the Sun AppletViewer.

## Debug on Demand



**If debug on demand is enabled, the debugger will be started:**

- If an unhandled exception occurs
- If an unrecoverable error occurs
- Start debug on demand
  - Enter `idod idebug` on a command line
- Stop debug on demand
  - Enter `idod /u` on a command line

### **Advantages of using debug on demand**

- The program can run at full speed, there are no delays caused by the debugger
- You don't need recreate error situations, when the programs produces an error, the debugger will start immediately

*Figure 9-19. Debug on Demand*

The debugger can be started in a way that it only starts intercepting the application code on certain conditions.

## Breakpoints



### Breakpoints supported by the DistributedDebugger

- **Line breakpoints** are triggered before the code at a particular line in a program is executed
- **Function breakpoints** are triggered when the function they apply to is entered (only for C/C++)
- **Method breakpoints** are triggered when the method they apply to is reached
- **Storage change breakpoints** are triggered when storage at a specified address is changed (not available on AIX, not for interpreted Java)
- **Load occurrence breakpoints** are triggered when a DLL is loaded. (not for interpreted Java)
- **Address breakpoints** are triggered before the disassembly instruction at a particular address is executed (not for interpreted Java)

*Figure 9-20. Breakpoints*

An elaborate breakpoint system allows to intercept an application on very specific conditions.

## Setting Line Breakpoints (1)

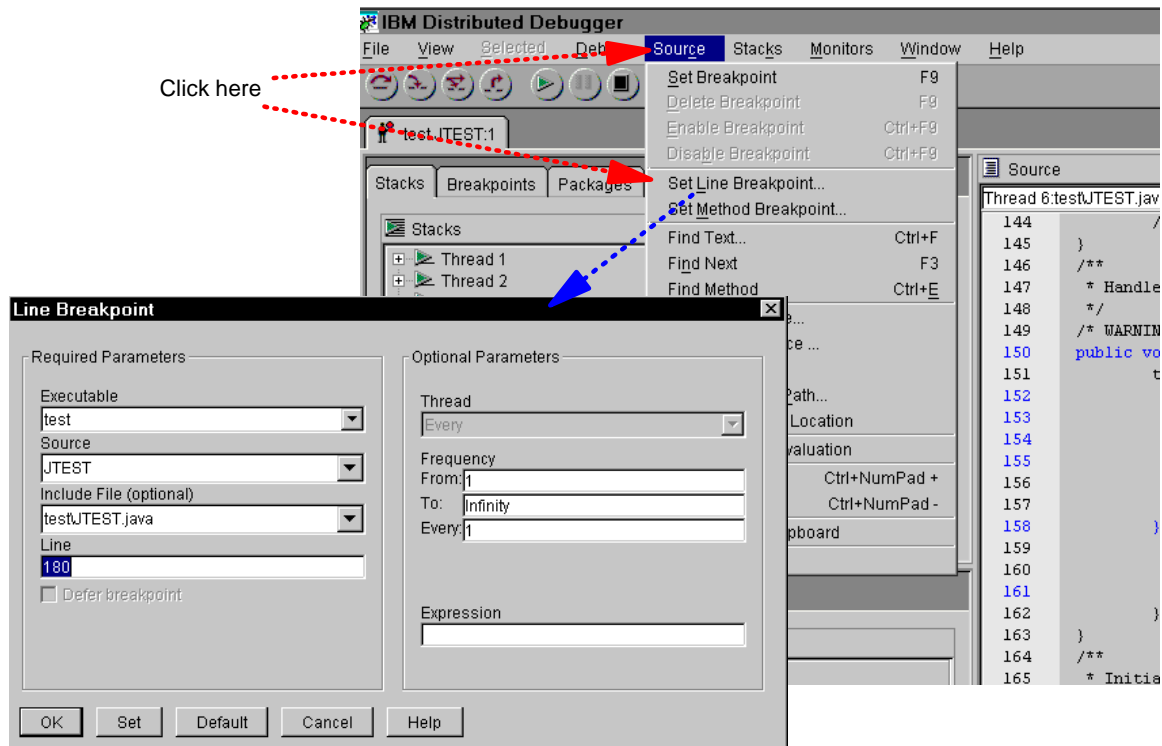


**Line breakpoints can be set from the source pane, the menu Breakpoint or the menu Source**

- Set a line breakpoint from the source pane
  - Select menu Source > Source view
  - Double-click on the line number in the prefix area or click right mouse button on the line and select Set Breakpoint
- Set a breakpoint from the Breakpoint menu
  - Select menu Breakpoints > Set Line
  - Fill out the dialog box with module name, executables, sources and so on
- Set a breakpoint from the Source menu
  - Select menu Source > Set Line Breakpoints
  - Fill out the dialog box with executable, include files, source and line number

*Figure 9-21. Setting Line Breakpoints*

## Setting Line Breakpoints (2)



VA Java V3 - Compiler/Debugger

© 1999 IBM Corporation

99SWB402UW

Figure 9-22. Setting Line Breakpoints (2)

## Setting Method Breakpoints (1)



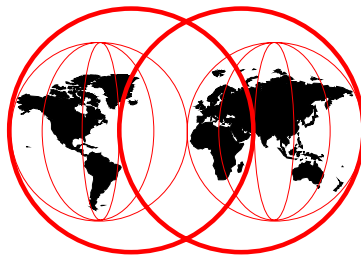
**Method breakpoints can be set from the Packages pane, the menu Source and the Menu Breakpoints**

- Set the method breakpoint from the Packages Panes
  - Expand the list in the packages pane until you see the method
  - right-click on that method, select Set Method Breakpoint
- Setting Methods from the Source menu and the Breakpoints menu is similar to the other breakpoints

*Figure 9-23. Setting Method Breakpoints*

# VisualAge for Java Version 3

Enhanced HPJ / Distributed Debugger



© 1999 IBM Corporation

99SWB402UW

*Figure 9-24. Setting a Method Breakpoint (2)*



## View/Modify/Enable/Disable Breakpts.



### Viewing breakpoints

- go to the Breakpoints pane
- Expand/collapse the list to see/hide breakpoints

### Modify breakpoints

- Click right on the breakpoint you want to modify
- Select Modify Breakpoint, and set properties
  - Which threads the breakpoint applies to.
  - How often the debugger should skip the breakpoint (the frequency).
  - Whether to stop on the breakpoint only when a given expression is true.
  - Whether to defer the breakpoint. Only line, function, or method breakpoints can be deferred.

### Enable/disable breakpoints

- Right click on a breakpoint in the Breakpoints pane
- Select Enable Breakpoint or Disable Breakpoint
- For enabling/disabling all breakpoints in your code, select menu Breakpoints -> Enable/Disable All Breakpoints

*Figure 9-25. View/Modify/Enable/Disable Breakpoints (1)*

## View/Modify/Enable/Disable BPs (2)



The screenshot shows the IBM Distributed Debugger interface. The main window displays the source code of `com.ibm.ivj.examples.vc.beandemo.BeanDemo:1`. The **Breakpoints** tab is active, showing a list of line breakpoints. A context menu is open over the first breakpoint, with options: **Disable Breakpoint**, **Delete Breakpoint**, **Modify Breakpoint...**, and **Breakpoint Properties...**. The **Line Breakpoint** dialog is open, showing the required parameters for the selected breakpoint. The **Properties for Source** dialog is also open, showing the properties for the source file.

**Line Breakpoint Dialog:**

- Required Parameters:**
  - Executable: `com.ibm.ivj.examples.vc.beandemo`
  - Source: `BeanDemo`
  - Include File (optional): `C:\MZIEGER\com\ibm\ivj\examples\vc\bea...`
  - Line: `12`
  - ☐ Defer breakpoint
- Optional Parameters:**
  - Thread: `Every`
  - Frequency: `From: 1 To: Infinity Every: 1`
  - Expression: (empty)

**Properties for Source Dialog:**

Property	Value
Package	<code>com.ibm.ivj.examples.vc.be...</code>
Class	<code>BeanDemo</code>
Source	<code>C:\MZIEGER\com\ibm\ivj\ex...</code>
Method Name	<code>&lt;init&gt;()</code>
Line	<code>12</code>
Address	
State	<code>Enabled</code>
Status	<code>Active</code>
Thread	<code>Every</code>
Conditional Expression	
From	<code>1</code>
To	<code>Infinity</code>
Every	<code>1</code>

VA Java V3 - Compiler/Debugger

© 1999 IBM Corporation

99SWB402UW









Figure 9-26. View/Modify/Enable/Disable Breakpoints(2)

## Program Execution Control



**Program Execution control is used for running, step into, step over, halting or stopping a program**



- all controllers are available as Icon, in the Debug menu and as shortcut
- Run a program  , Run, F5
- Step Over  , Step Over, F10
- Step Into  , Step Into, F11
- Step Debug  , Step Debug, F7
- Step Return  , Step Return, shift + F11
- Halt Program  , Halt, ctrl + F5
- Terminate Exec.  , Terminate, shift +F5
- Restart prgrm.  , Restart, ctrl + shift + F5

*Figure 9-27. Program Execution Control*

A tool bar with buttons controls the execution of the program code.

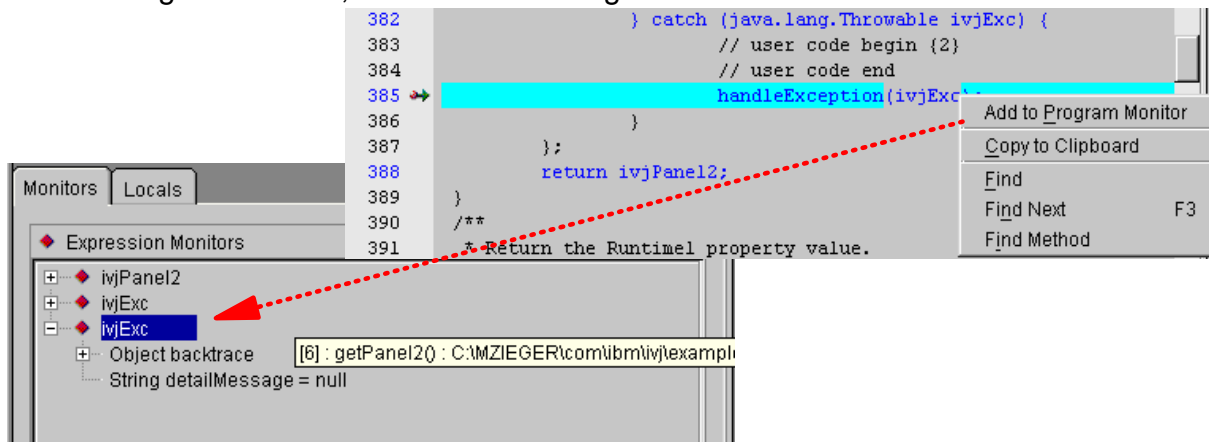
You can step into code, step over, etc.

## Inspecting Data



**For interpreted Java code, you can watch variables.  
For all compiled native code, you can also watch registers and storage**

- Adding a variable to watch
  - Highlight the variable you want to watch in the source pane
  - Right-click on it, select Add To Program Monitor



VA Java V3 - Compiler/Debugger

© 1999 IBM Corporation

99SWB402UW

Figure 9-28. Inspecting Data

## Object Level Trace (OLT)



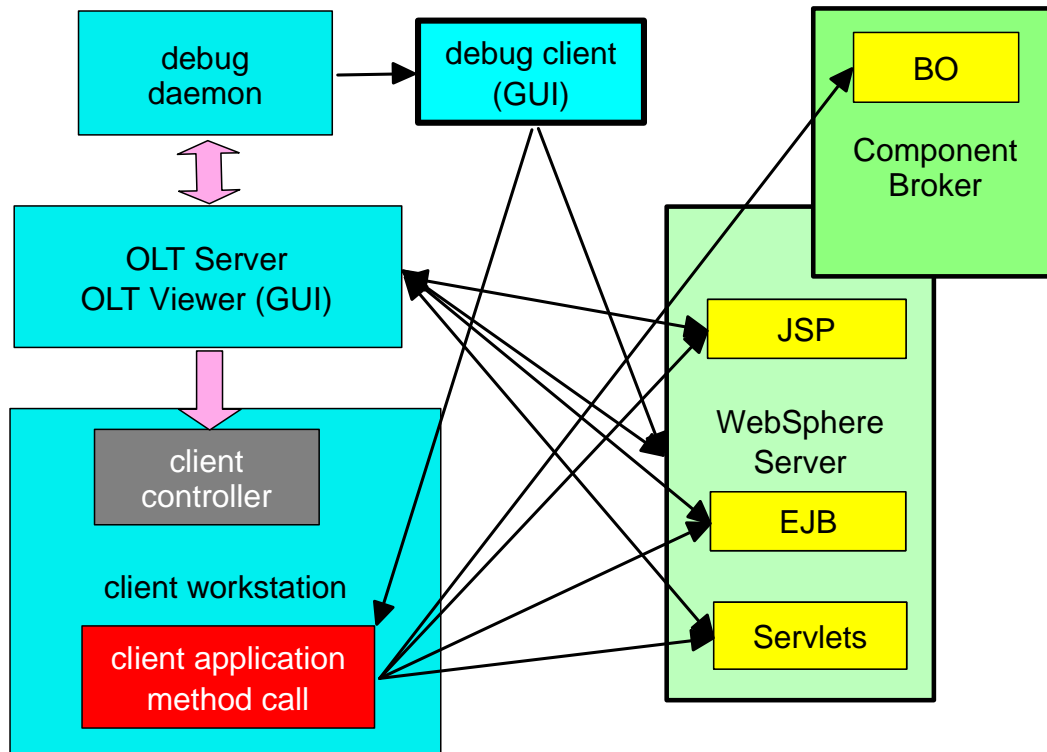
### OLT is an extension to the distributed debugger

- Trace and debug multilingual distributed applications from one workstation
- Supported languages
  - As a server: C++ on AIX and NT, Java on AIX and NT
  - As a client: C++ on AIX and NT, Java on AIX and NT
- Supported models
  - As a server: C++ BO, Java BO, EJB, JSP, Servlets
    - BO: BusinessObjects from IBM Component Broker
  - As a Client: Java, C++, ActiveX
- Not all combinations from the components are possible

Trace program flow across client/server

*Figure 9-29. Object Level Trace (OLT)*

## OLT Schema



VA Java V3 - Compiler/Debugger

© 1999 IBM Corporation

99SWB402UW

Figure 9-30. OLT Schema

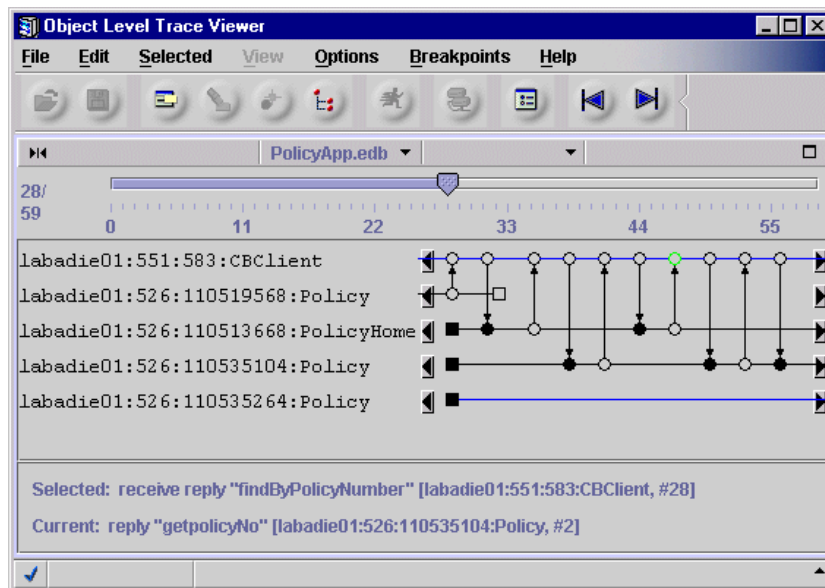
## OLT Step-by-step



- Compile your application code for OLT
  - CB: use the IVB\_TRACE\_DEBUG option
  - WS: compile with javac -g option
- Enable OLT for Component Broker or WebSphere
- Tracing the distributed application
  - Start OLT (enter OLT on a command line)
  - Start your C++ or Java application
- Setting method breakpoints in the trace
  - Method breakpoints can be only set to debuggable server events
  - These events are marked by a filled circle in the trace
  - Click-right on the circle, select Add to method breakpoint list
- Running the Debugger with OLT
  - Start OLT and select the Online Mode
  - Deselect menu Options > Step-by-step debug mode
  - Select Trace and debug with prompt, on the Monitoring Mode page
  - Start your client application

*Figure 9-31. OLT Step-by-Step*

## Sample OLT Trace



- Trace file can be saved/loaded
- Navigation shortcuts helping to step through a trace

Figure 9-32. Sample OLT Trace



## Summary



### High-Performance Java Compiler

- Part of the Enterprise Toolkit of VA Java 3.0 EE

### Distributed Debugger

- Installation
- We have learned, how to debug local and remote applications
- Working with breakpoints is easy
- Navigate through the code with the debugger
- Inspecting and changing expression values

### Object Level Trace

- We learned the concept of OLT
- We know, how to setup an environment for debugging distributed application
- OLT is a tool for creating and reading traces of distributed applications

*Figure 9-33. Summary*

The High-Performance Compiler is still there, doing an even better job.

The Distributed Debugger has been enhanced and supports distributed multi-lingual applications with the Object Level Trace facility.

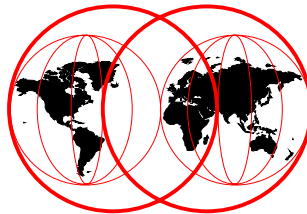


# 10

## VA Java Version 3 Domino Access Builder

**VisualAge for Java Version 3**

Domino Access Builder



© 1999 IBM Corporation

99SWB402UW

## Objectives



### Introduction to Notes access from Java

- Accessing notes 2 ways
- IIOP and local access
- Building applets, agents, separate applications

### Domino Access Builder

- Features
- Requirements
- Setting up standalone and with Notes client
- Using the generic beans: create connection, make a database
- Generating beans
  - SmartGuide with access to Notes
- Deploying Notes Java applications
  - standalone, with Notes client
- Attention

*Figure 10-2. Objectives*

VisualAge for Java Version 3 provides improved access to Domino/Notes from Java.

Access is now provided through IIOP, directly to the server.

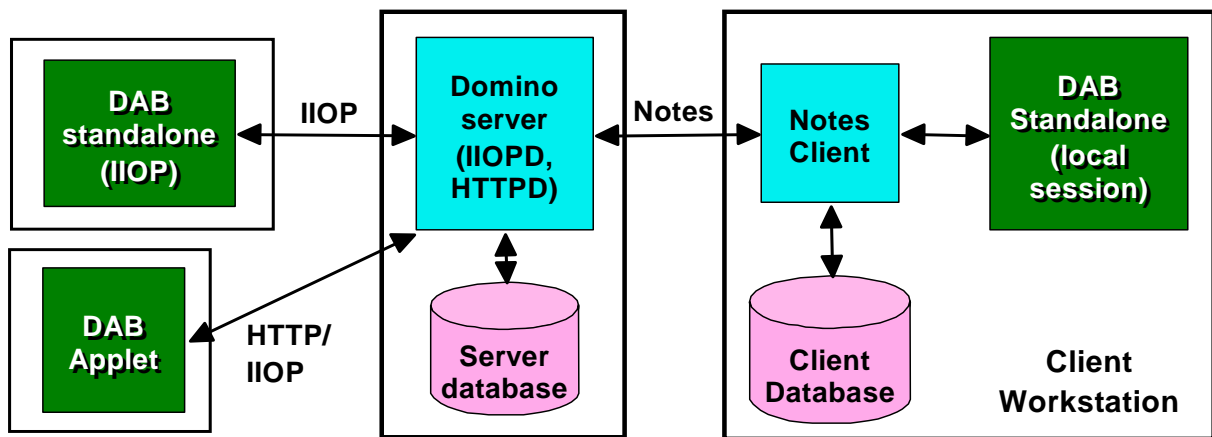
Programming is simplified by JavaBeans and by the Domino Access Builder.

# Domino Access Builder Introduction



## What is Domino Access Builder?

- Beans for use in the Visual Composition Editor
- Uses the Lotus Domino Libraries to access Notes Databases
  - Standalone to server - without local Notes code installed
  - With a local installation of the Notes client or server



VA Java V3 - Domino Access Builder

© 1999 IBM Corporation

99SWB402UW

Figure 10-3. Domino Access Builder Introduction

The Domino Access Builder uses the Lotus Domino Libraries to access the Notes databases.

Depending on the type of connection 2 different sub-libraries will be used.

For a local session the Notes.jar library is used, using the JNI to get access to local Notes binaries.

When using IIOP an all new sub-library NCSO.jar is used, that is independent from any local Notes binaries.

## Domino Access Builder Features



### Beans on top of the Lotus Domino Java Libraries

- Generic beans
  - wrapper classes for databases, forms, views, and other Domino design elements
- Generated beans
  - SmartGuide for generation of beans
  - Generated beans have a 1:1 relationship from attributes to form content
- Transparent session management
- Easy-to-use error handling facility

### New in the Lotus Domino Java Libraries

- Access a Lotus Domino Server from a standalone Java application via IIOP - not using any local Notes installation

VA Java V3 - Domino Access Builder

© 1999 IBM Corporation

99SWB402UW

*Figure 10-4. Domino Access Builder Features*

With Version 3, instead of using the Lotus Domino Java libraries, it is now possible - but not necessary - to use the VCE for development.

It is not necessary to use the complex Lotus Domino Java Libraries for most development.

There is now - using the version 5.0 libraries and server - the possibility of accessing the databases without having the Notes client or server coed installed.

# Requirements



## Development

- VA Java version 3.0
- Local Notes client/server installed or
- Network access to an Lotus Domino Server R 5.0 with HTTPD and IIOp enabled

## Deployment

- Local Notes client/server installed
  - jar files ablib30.jar and Notes.jar must be included in the class path of the application
- Network access to an Lotus Domino Server R 5.0 with HTTPD and IIOp enabled
  - jar files ablib30.jar and NCSO.jar must be included in the class path of the application

*Figure 10-5. Requirements*

The two ways of access require different Jar files when deployed.

Within VA Java you can work with the projects or with the Jar files.

Deployment: if it is an applet the class must also include the NCSO.jar files and the ablib30.jar files like this:

```
<applet code="com.ibm.ivj.domino.ab.samples.todo.ToDoSendList.class"
width="900" height="550" ARCHIVE="ablib30.jar,ncsoc.jar,swingall.jar">
```

## Installation



### Prepare workspace

- Remove the IBM WebSphere Test Environment project (or the RMI\_IIOP project)
  - contains com.ibm.CORBA.iio
  - conflict with Domino Java library
- Add the necessary libraries
  - **File->Quick Start->Features->Add feature**
    - Select the Domino Access Builder Libraries 3.0 feature
    - This also loads the Lotus Domino Java libraries 5.0 feature
- Setup up classes or workspace for execution and debugging
  - Domino Access Builder Library (eab\runtime30\domino\ab\ablib30.jar)
  - Lotus Domino Java library 5.0 (imported from Notes.jar and NCSO.jar)
- or add Jar files to class path

Figure 10-6. Installation

The reason for removing the WebSphere or IBM RMI\_IIOP project is that it contains some of the same packages as the Lotus Domino Java Library and the workspace does not allow multiple projects to refer to the same packages.

The developers of VA Java state that because of integration issues it was decided to include the same packages twice (actually three times), which is why one has to remove the other projects.

Notes.Jar and NCSO.Jar are lotus domino files supplied with the Lotus Notes Client and Server. They contain the JNI classes for accessing the local Notes client binaries respectively the classes (no JNI) to access a separate Lotus Domino server using the IIOP protocol. They are also distributed in a way with VAJ, the Lotus Domino Java Libraries was created by importing the Notes.jar file and the NCOS.jar file into VAJ.

Notes.jar can be found in x:\notes or x:\lotus\domino

NCSO.jar can be found in x:\notes\data\domino\java\NCSO.jar or x:\lotus\domino\data\domino\java\NCSO.jar



# Workspace Configuration



## Setup up workspace for test and debugging

- Either setup the class classpath to include the 2 projects added to the workspace
- Or extend the workspace class path
  - with the ablib30.jar
  - and NCSO.jar for IIOP access to databases
  - and Notes.jar for local session access to database

• Do not mix the two approaches - it does not seem to work

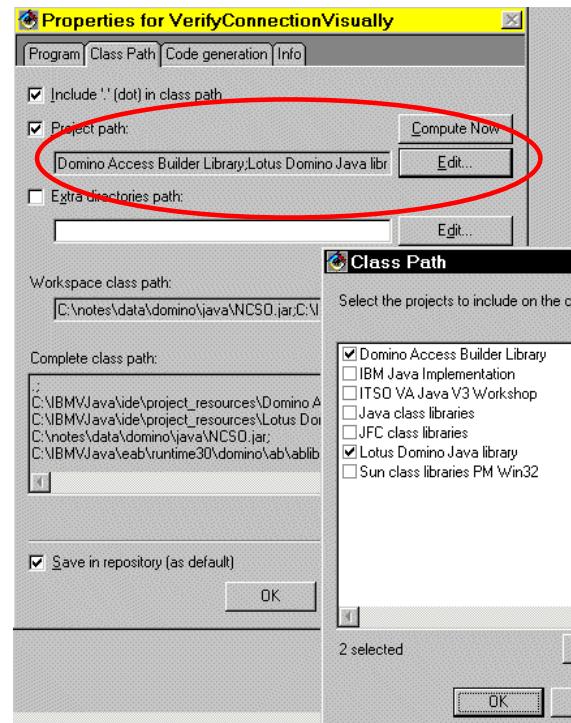


Figure 10-7. Workspace Configuration

The main reason for using the class path of the workspace instead of the class, is convenience. The goal is to avoid having to add the 2 projects to all main classes used for DAB development (it is of course only classes that will be the starting point of an application that require the 2 projects added).

Funny thing is though that using the VCE the Swing class library and the DAB class library will be added to the class class path. But not the Lotus Domino Java library. This can give cause to some problems when executing the Java application inside the IDE.

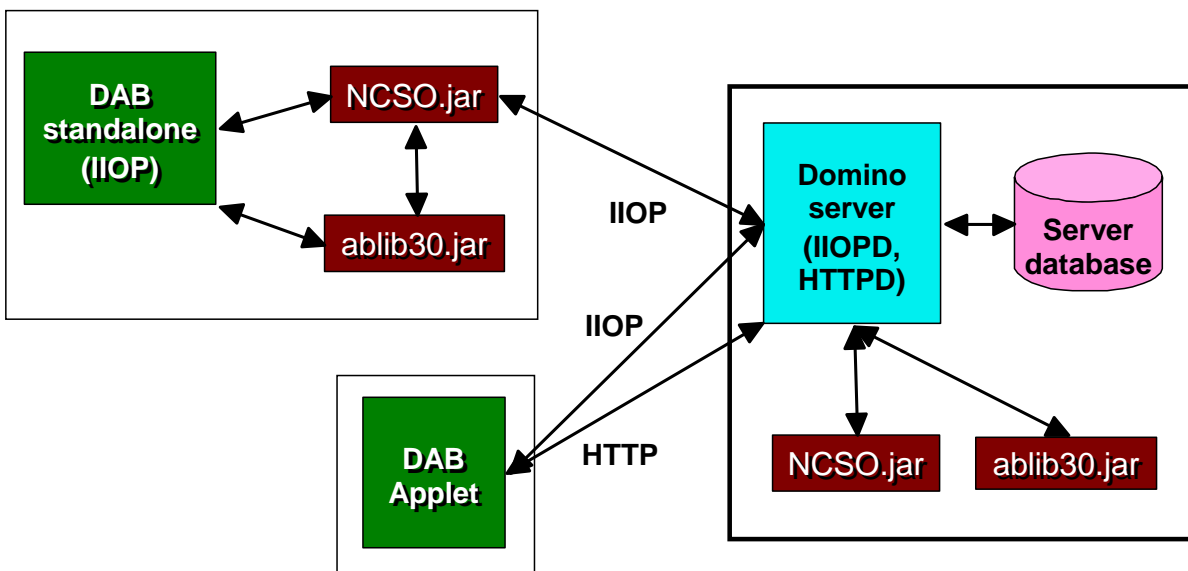
For the IDE to be able to execute any java class that uses the DAB all classes must be loaded either by the workspace class path or by the class class path. That is either the class class path must include both Lotus Domino Java Libraries and the Domino Access Builder library or the workspace class path must include NCSO.jar and ablib30.jar. Having one included in the class class path and the other included in the workspace class path will cause ClassCastExceptions when accessing a NotesDocumentCollection. As the VCE adds the Domino Access builder library, this will often cause problems.

Experience has also shown the it is a bad idea to export the Lotus Domino Java Libraries into a jar file and use that, as this will generate an exception of the type ExceptionInInitializerException.

## Application Connections



Depending on the standalone application type (applet or application) the jar files must either be present on the client or the server



VA Java V3 - Domino Access Builder

© 1999 IBM Corporation

99SWB402UW

Figure 10-8. Application Connections

When connecting to a server without a Notes client, then the Jar files are required for applications.

For applets the Jar files must be at the server for automatic download.

## Creating a Connection Visually



### To connect to a notes server you need

- Connection specification bean (NConnectionSpec)
- Logon specification bean (NLogonSpec)
- Session bean (NSession)  
contains connection spec bean and logon bean

### Setup connection information

- Connection type
  - Local session (server or client)
  - IIOP (server)
  - Applet (server)
- Server info
  - dependant on connection type
- Logon information

Domino Beans

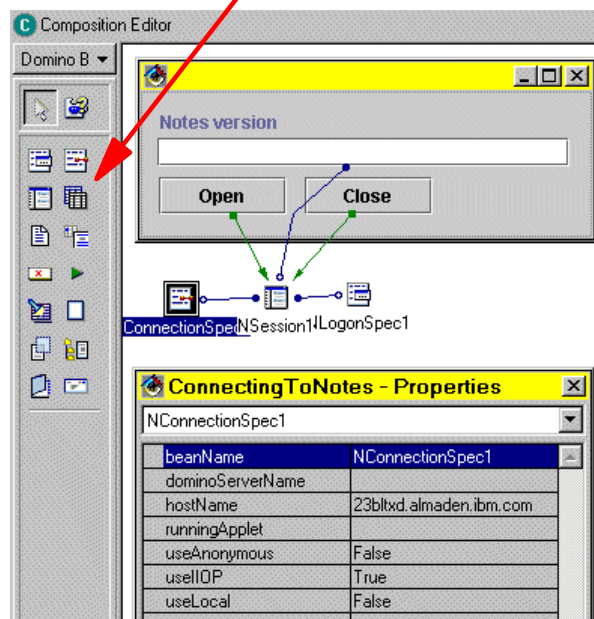


Figure 10-9. Creating a Connection Visually

The session bean is the one controlling the connection. The logon spec and the connection spec contain data for enabling the session. The Logon and Connection beans can also be torn-off the Session bean, which requires the attributes (userid, server name etc.) to be set dynamically (runtime).

The Open button is connected to the open method of the Session bean, the close is connected to the close method. The Notes version property is connected to the text property of the Text Field bean.

If you use a local Notes client, then the server name is empty, and no userid/password are required.

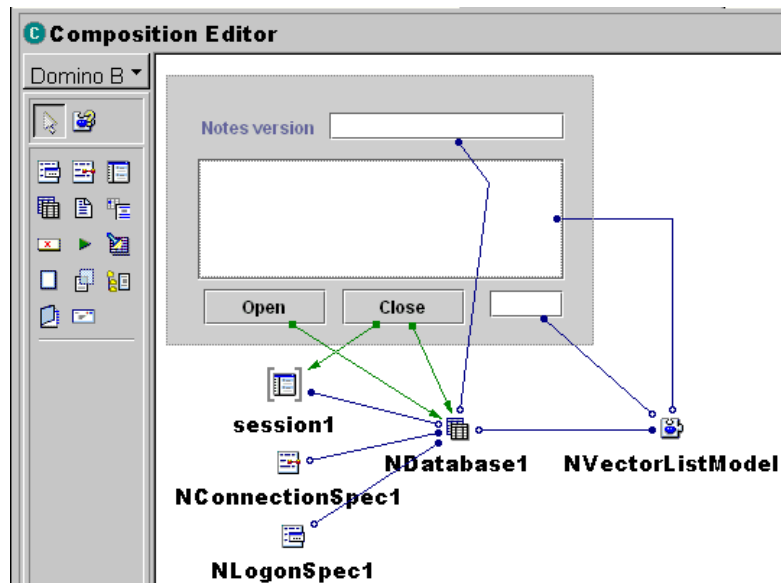
## Fetching Data Visually (Simple)



To retrieve a list of all documents from a notes database you need:

- A database bean
- A connection specification bean (NConnection)
- A logon specification bean (NLogon)
- A VectorListModel bean (NVectorListModel)

Note: db.close does not close session



VA Java V3 - Domino Access Builder

© 1999 IBM Corporation

99SWB402UW

Figure 10-10. Fetching Data (Simple)

With just a few connections you can see the documents in a database.

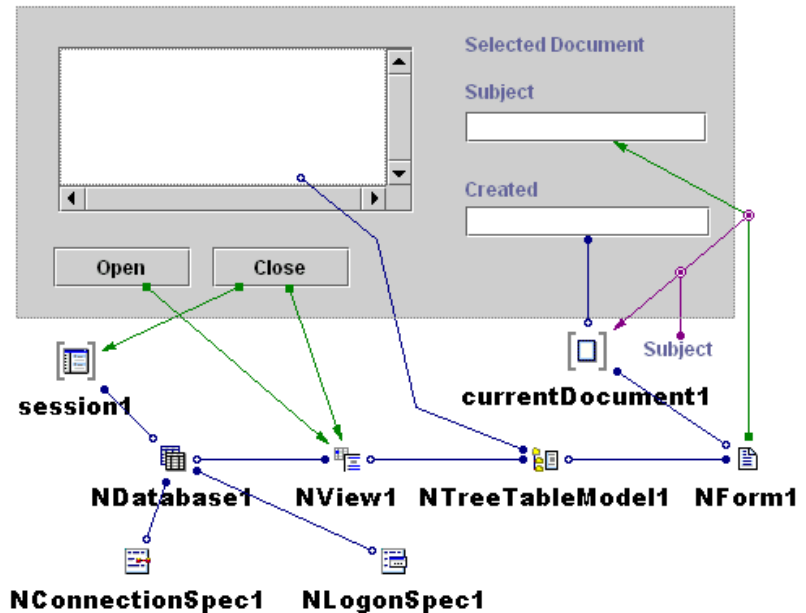
Note that documents will not be formatted with such simple logic.

## Fetching Data Visually (Complex)



### A more complex example of retrieving data:

- A database bean
- A connection bean
- A logon bean
- A view Bean (NView)
- A TreeTableModel bean (NTreeTableModel)
- A Form bean (NForm)
- A GUI bean called NTreeTable
- A document bean (currentDocument)



VA Java V3 - Domino Access Builder

© 1999 IBM Corporation

99SWB402UW

Figure 10-11. Fetching Data Visually (Complex)

The Open and close buttons as usual - almost. They are connected to the NView bean, which is again connected to the NDatabase bean. The thing to know is that the NView bean will open and close the underlying database bean, when the open and close methods of the NView bean are used. The Model property of the NTreeTable bean is connected to the NTreeTableModel bean.

The NTreeTable is a very special bean combining the JTree and JTable beans, emulating the appearance (aka categories) of a Notes View. The NTreeTable bean is inside a JScrollPane bean. Connect the NTreeTable(this) to the treeModel of NTreeTableModel.

To get document specific information, connect the currentDocument properties of the table model and the form. Tear-off the currentDocument from the form and connect its properties to the GUI.

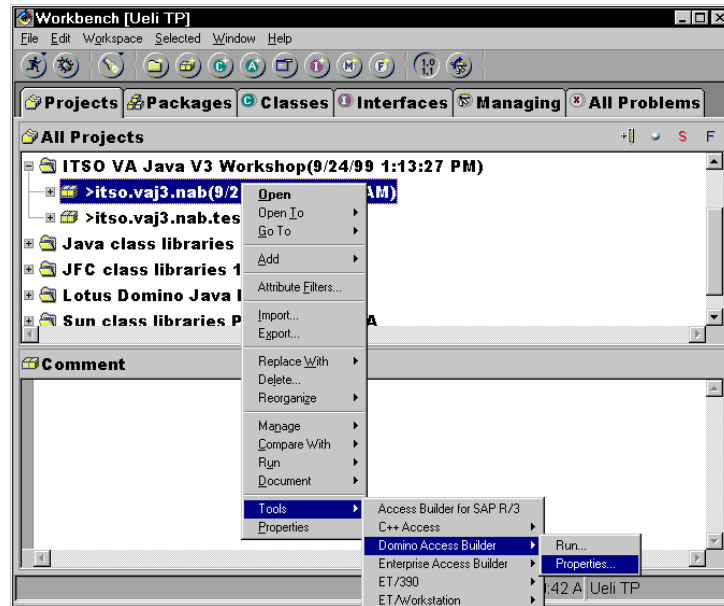
A small trick is used to get a specific item from the currentDocument to the text field. The Form bean currentDocument event is connected to the text property of the text field with the getItemValue as the argument. getItemValue requires the item name as input to retrieve the value. This is done by creating a Label bean outside the visible area, with the text property set to the item name requested. The text property is then connected as the input to the getItemValue method of the Document bean and Hokus-Pokus - a document item is retrieved visually - no strings (code by hand) :-).

## Generating Domino Beans



### Select the package where the beans go

- Right click with mouse
- Select *Tools->Domino Access Builder*
- Select *Properties* to set up the SmartGuide (only necessary to configure once)
- Select *Run* to start the SmartGuide to generate the Domino Beans



VA Java V3 - Domino Access Builder

© 1999 IBM Corporation

99SWB402UW

Figure 10-12. Generating Domino Beans

Start the Domino Access Builder from the Tools action. First the properties should be set.

## Setup SmartGuide to Generate Beans



### To use the Domino Access Builder SmartGuide

- Setup the properties
  - Connection type (IIOP)
  - Connection information

Domino Access Builder Properties SmartGuide

Server Selection

Enter TCP/IP server name

Server name: 23bltd.almaden.ibm.com

Select user ID and password options:

☐ Supply user ID and password now

User ID:

Password:

Confirm:

☒ Prompt when needed

Test connection:

< Back  Finish Cancel

VA Java V3 - Domino Access Builder © 1999 IBM Corporation 99SWB402UW

Figure 10-13. Setup SmartGuide to Generate Beans

With the properties of the Domino Access Builder you define the connection to the server (or local client).

This is actually the easiest way to check if the Domino Access Builder feature is working. After selecting the connection type and configuring the connection information, click the *Test* button to connect to the server/local client as specified. If it works, it should work from any class within VA Java.

## Select Database



Select which database (file) to access, using the directory browser

- The directory browser accesses the Data directory of the server/local client the SmartGuide has been setup to connect to

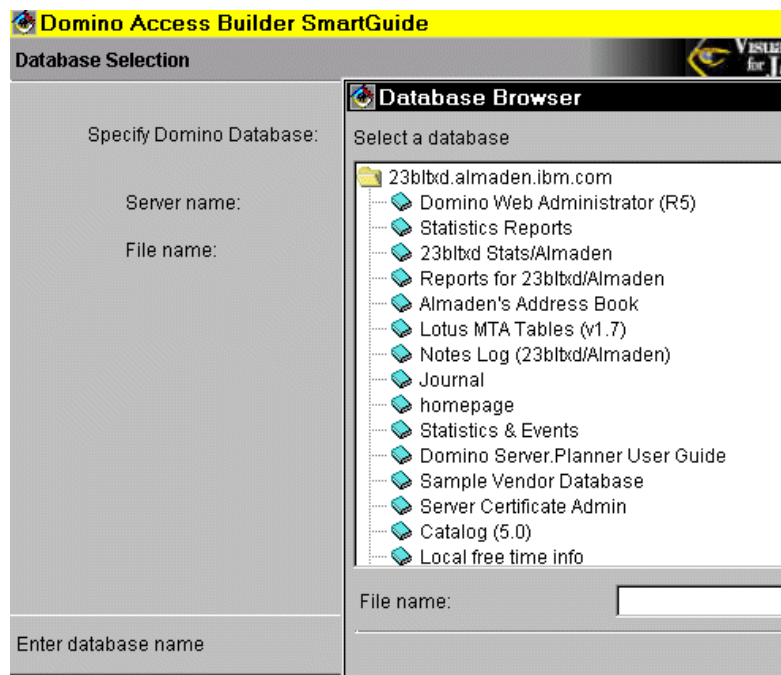


Figure 10-14. Select Database

The Database Browser lists all the databases for your selection.



## Select the Forms and Fields



For each form selected, the fields to include in the generated beans must be specified

### Default:

- no fields are included

Field Name	Field Type
Body	Text
FolderOptions	Text
SavedOnce	Text
Subject	Text
TimeCreated	Text

VA Java V3 - Domino Access Builder

© 1999 IBM Corporation

99SWB402UW

Figure 10-15. Select the Forms and Fields

This is opposed to the user guide manual, which states that all fields by default are included.

## Complete the Generation of Beans



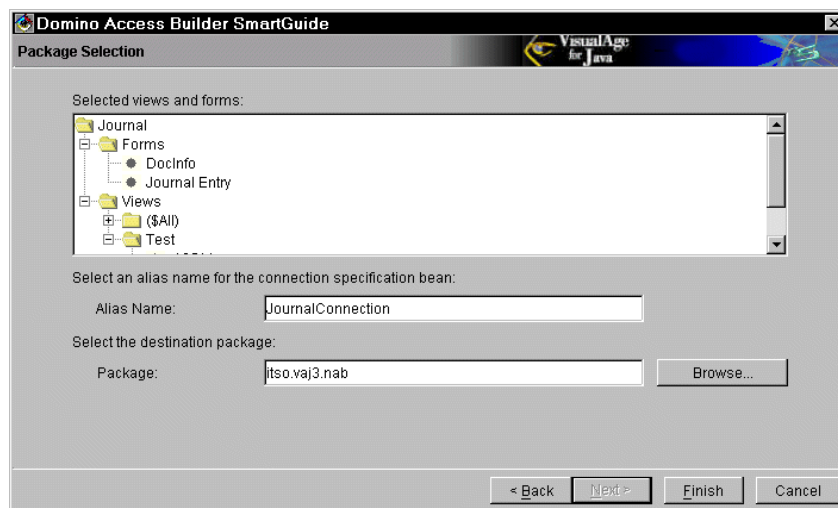
Select the views to beans generated

The final page in the Domino Access Builder SmartGuide gives an overview of the selections made

- Database
- Forms
- Fields (check this!)
- Views

It is possible to

- select an alias for the Connection Specification bean
- Change the package name



VA Java V3 - Domino Access Builder

© 1999 IBM Corporation

99SWB402UW

Figure 10-16. Complete the Generation of Beans

If the package do not exists it will be created. Please note that the generated beans are created as edition only and not versioned.

## Generated Beans



**For the Journal DB, Test View, the SmartGuide generates:**

- TestView
  - main entry, open here
  - includes link to a connection bean
- JournalNSFConnection
  - tailored ConnectionSpec bean
- Journal\_EntryForm
  - the form selected
  - through the view we retrieve a collection of forms
- Journal\_EntryDocument
  - tailored document bean with properties for the selected fields
- Journal\_EntryFormQuickForm
  - Swing bean to display the attributes of the document

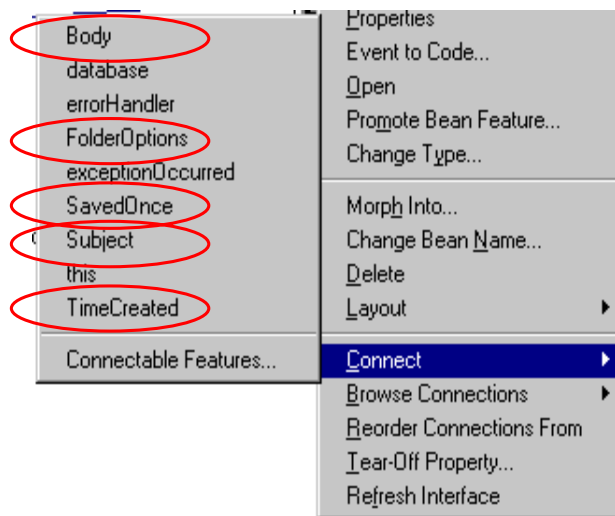
*Figure 10-17. Generated Beans*

## Using Generated Beans



**This basically works as previously described, but ...**

- The generated beans must be added manually to the bean palette or the VCE
- All the fields (items) included in the SmartGuide, when the beans where generated, are now properties in the generated document beans

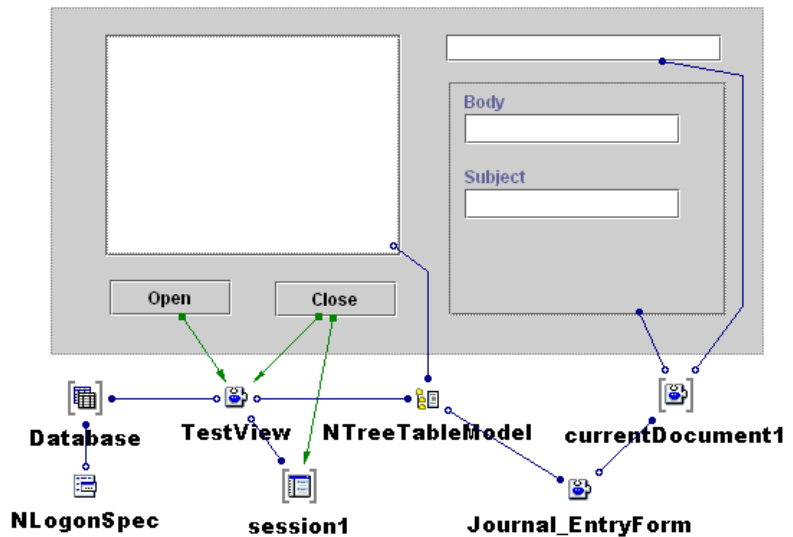


*Figure 10-18. Using Generated Beans*

## Visual Comp. with Generated Beans



- Generated View bean is opened
  - has connectionSpec inside
- NTreeTableModel has the documents
- currentDocument becomes the form
- currentDocument of form is connected to the FormQuickForm



VA Java V3 - Domino Access Builder

© 1999 IBM Corporation

99SWB402UW

Figure 10-19. Using Generated Beans

This is similar to the example with the standard beans.

Because we are using tailored beans, the document properties are now the actual documents of the view.

The table model holds a collection of journal entry documents. The currentDocument that is extracted can be connected to the tailored GUI QuickForm (document property). The attributes are displayed in the GUI.

## Careful with Local Sessions



- Do not run other Domino applications like the Notes client, while developing applications.
- Never terminate a thread that accessed a Domino Access Builder bean during a debug session. Do not use `thread.stop()`. Never terminate an application using the debugger or the console.
- Always add the following lines to terminate your code:
  - `NSession.closeOpenSessions();`
  - `System.exit(0);`
- If application code throws an exception that cannot be resumed, exit VisualAge for Java and restart it.
- Do not use `java.lang.Thread` objects that access Domino Access Builder objects. Use `com.ibm.ivj.domino.ab.model.NThread` objects.
- Do not terminate `NThreads` through `Thread.stop()` or from the debugger once they have accessed a Domino Access Builder bean.
- In some situations, additional threads other than the AWT event dispatch thread are created (for example JFC). These threads are not allowed to access Domino Access builder beans.
- Do not run two applications that access Domino Access Builder methods at the same time from within the IDE.

Figure 10-20. Careful with Local Sessions

In the examples shown we always connected the *Close* button to the *closeOpenSessions* methods of the session. The session is extracted from the database or view bean.

## Summary



### Notes Java Classes

- access to Notes using IIOP
- with or without local client

### Domino Access Builder

- creates powerful beans for Notes applications
- generic beans
- tailored beans for a database form

*Figure 10-21. Summary*

The Domino Access Builder provides powerful beans that make Notes programming easy.

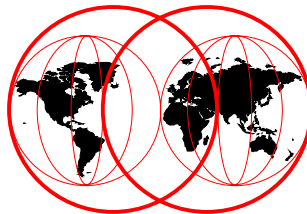




# 11 **VA Java Version 3 DB2 Stored Procedure Builder**

**VisualAge for Java Version 3**

DB2 Stored Procedure Builder



## Objectives



### Stored Procedures

- What is new about Stored Procedures in DB2 V. 6.1

### Stored Procedure Builder

- Features
- Requirements

### Using Stored Procedure Builder

- Setup
- Create a Stored Procedure

### Integration with VA Java

Stored Procedures are not standardized across DBMS vendors

*Figure 11-2. Objectives*

The DB2 Stored Procedure Builder requires DB2 6.1.

## Stored Procedures in DB2 V 6.1



### New features of Java interface

- Support for the SQLJ Routines Specification
- Out and InOut parameters can now be single element Java arrays
  - Not necessary to use the `set(parm_num, value)` method to return values
- New `CREATE PROCEDURE` options
- Stored Procedure can be installed as a jar file
- No mandatory inherited class
  - Stored Procedure no longer needs extends `StoredProc`
- Java Stored Procedure now maps to `static void` method
- Support for storing the Java source code at the database
- Support for remote, source level debugging of Java Stored Procedure

*Figure 11-3. Stored Procedures in DB2 V 6.1*

Stored procedure can be written in Java for quite a while.

DB2 6.1 provides a number of improvements that make it easier to write stored procedures in Java.

## Installing Stored Procedures



### From Java source:

**==> DB2 Catalog Tables**

```
javac IncSalaryStp.java
copy IncSalaryStp.class c:\sqllib\function
db2 CREATE PROCEDURE userid.IncSalaryStp
    (in increase int, in empno char(6)) language java
    parameter style java not fenced external name
    'IncSalaryStp!execute'
```

### From Jar file:

Jar-name	class-name	method-name
----------	------------	-------------

```
db2 call sqlj.install_jar
    (file:BIGSALARY.jar,Userid.BIGSALARY)
db2 CREATE PROCEDURE userid.BIGSALARY (in SALARY int)
    language java parameter style java
    dynamic result sets 1 fenced external name
    'USERID.BIGSALARY:itso.vaj3.spb.BigSalary.bigSalary'
```

► Fenced: separate address space

Not-fenced: DB2 address space

Figure 11-4. Installing Stored Procedures

Stored procedure can be installed from class files or from jar files.

In both cases a CREATE PROCEDURE statement is executed to define the stored procedure to DB2.

A jar file is installed with CALL SQLJ.INSTALL\_JAR and it puts the code into two catalog tables.

With the Stored Procedure Builder all this is automated.

# VA Java and Stored Procedure Builder



## Stored Procedure Builder

- is part of DB2 V 6.1
- is a stand alone application, VA Java is not required
- can be accessed from within VA Java
- stores the stored procedures source code in VA Java
- is a Java application

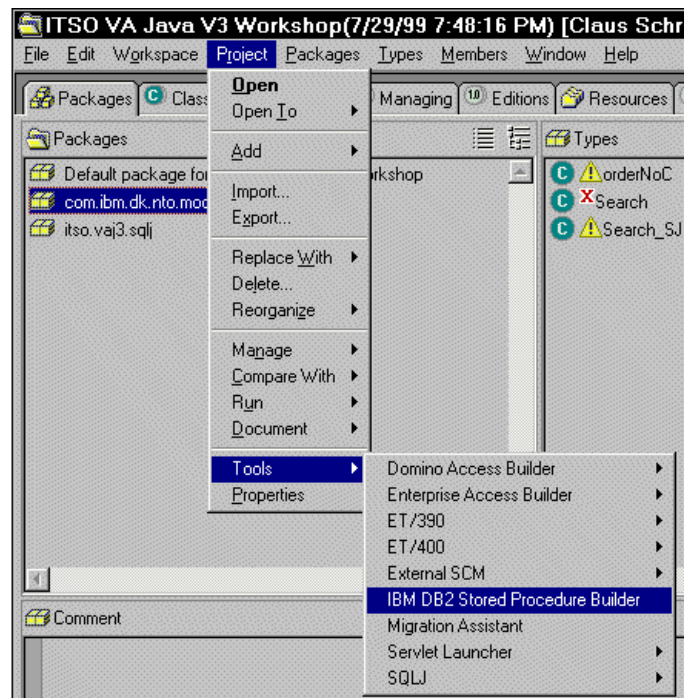


Figure 11-5. VA Java and Stored Procedure Builder

You can run the DB2 Stored Procedures Builder inside VA Java or stand-alone.

## Features of Stored Procedure Builder



- **Integration with popular IDEs**
  - VisualAge for Java
  - Others
- **Single tool supporting the entire DB2 family (over time)**
- **Develop in one place, deploy to remote DB2 servers**
- **Integration with IDE allows Java source code to be stored in IDE repository**
- **Workstation GUI development environment**
- **Support rapid/iterative development**
  - SmartGuides simplify/supports complex tasks
- **Initially only support for Java Stored Procedures**
  - Support for both JDBC and SQLJ
  - Support for SQL(PSM) Stored Procedures (Oracle) will follow later

*Figure 11-6. Features of Stored Procedure Builder*

# Requirements



## For the development client:

- One of the supported operating systems
  - Windows NT, 95, 98
  - Others to follow
- DB2 V 6.1 installed
  - VA Java requires Fixpack 1 + additional fixes (=> Fixpack 1a)

## For the deployment of Java Stored Procedures to servers

- Server operating system
  - NT, 95, 98, AIX, Sun Solaris, OS/2
  - Other platforms will follow later including OS/390 and OS/400

## Debugging

- Debugger client: Windows NT
- DB2 Server: Windows NT, AIX

Figure 11-7. Requirements

## Start Stored Procedure Builder



- First Select a project  
(the Stored Procedure Builder is only accessible in the project context)
- Setup the connection information  
(only first time)
  - Select the database
    - Create the alias if required
  - add userid, password
  - setup filter information if required, for the list of Stored Procedures

Figure 11-8. Start Stored Procedure Builder

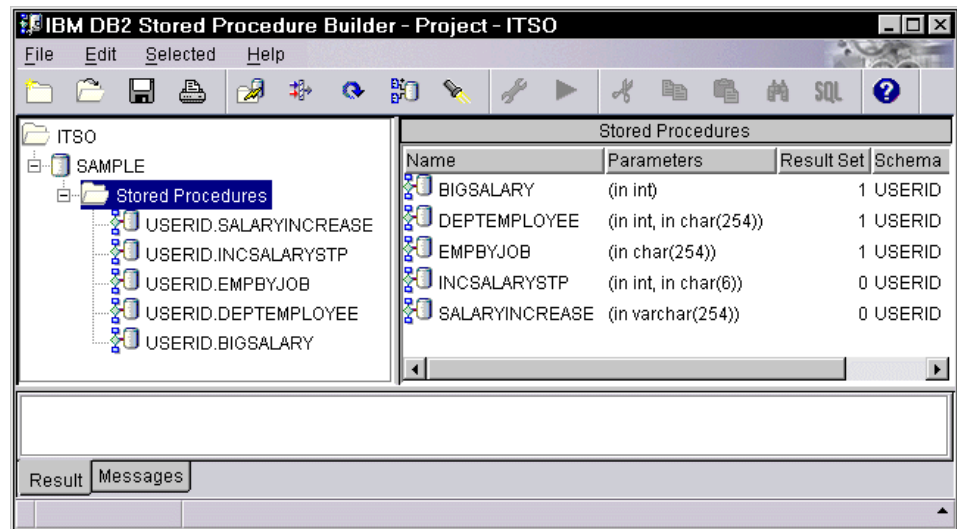


# Stored Procedure Builder GUI



## Project view provides:

- High level view of connections and Stored Procedures
- Launch point for all user tasks
- Results and messages area



VA Java V3 - DB2 Stored Procedure Builder

© 1999 IBM Corporation

99SWB402UW

Figure 11-9. Stored Procedure Builder GUI

## Configure Stored Procedure Builder



- **Environment properties**

- Connection
  - Can launch DB2 Client configuration Assistant
- Editor (tailor)
- Assistance (user guidance)
- Result (# rows, col width)
- Debug (tcp/ip port)
- SQL Types (view case)
- Type Mapping  
(String = varchar(254))

- **Project properties**

- Project name
- Description
- Copyright statement

- **Procedure properties**

- Stored procedure
- Parameters
- Options  
(Jar name, fenced)

Project is stored as ***projectname.spp*** file in  
VA Java Resources (or stand-alone file)

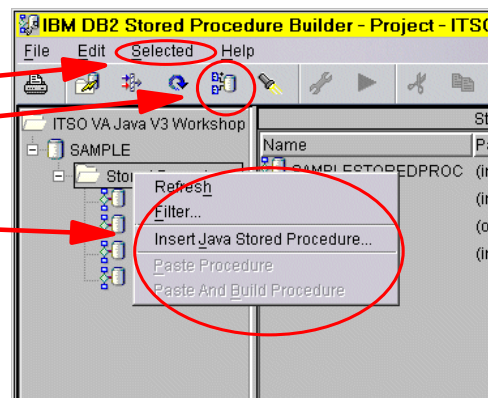
Figure 11-10. Configure Stored Procedure Builder

## Create the Stored Procedure



### Insert a new Java Stored Procedure

- Use menu-item
- Use toolbar
- Use pop-menu



### Define the parameters for the Stored Procedure

- Name (inclusive qualifier)
- Pattern (JDBC/SQLJ, other elements)
- Specify the query
- Optional Parameters (IN, InOut, OUT)
- Other options (package name, build process)

Figure 11-11. Create the Stored Procedure

## Code Generator Parameters



### Pattern

- The generated Stored Procedure will either contain a single query or a switch statement with a number of queries
- Select if the Stored Procedure should return the result set to the caller
- Errors can be handled 2 ways
  - Thrown as SQLExceptions
  - Caught and returned as output arguments

The screenshot shows a dialog box titled "Inserting Java Stored Procedure - STADE1.Proc" with four tabs: "1. Name", "2. Pattern", "3. SQL Query", and "4. Parameters". The "2. Pattern" tab is selected. The dialog is titled "Step 2 of 5: Pattern". Below the title, it says "Specify characteristics that describe the pattern of the stored p one query or choose among several. It can return a result set a". There are three sections: "Query", "Output", and "Errors". In the "Query" section, "Run a single query" is selected with a radio button. In the "Output" section, "Return a result set" is checked with a checkbox. In the "Errors" section, "Generate an SQLException if an SQL error occurs" is selected with a radio button. Below this, there is a section "Output arguments for:" with three checkboxes: "SQLSTATE", "SQLCODE", and "SQL message", all of which are currently unchecked.

Figure 11-12. Code Generator Parameters

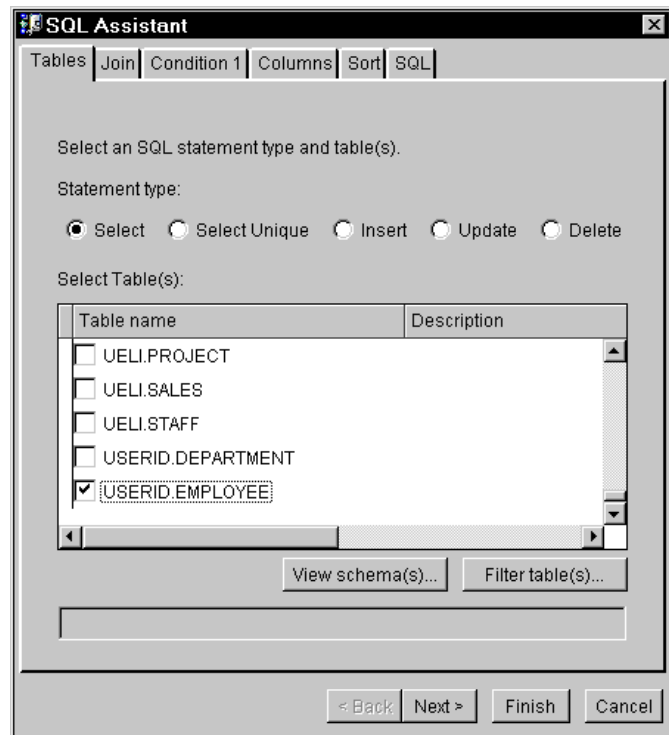
## Define the Query (one or more)



### SQL Assistant

- Select tables
- Create joins  
(Inner, left outer, right outer)
- Selection conditions  
(WHERE clause,  
host variables)
- Result columns
- Sort order
- Final SQL

**Very similar to  
Data Access Beans**



VA Java V3 - DB2 Stored Procedure Builder

© 1999 IBM Corporation

99SWB402UW

Figure 11-13. Define the Query (one or more)

## Conditions and Generated SQL



The image shows two screenshots from the VisualAge for Java IDE. The left screenshot is the 'SQL Assistant' dialog box, which is used to define conditions for a stored procedure. It has tabs for 'Tables', 'Join', 'Condition 1', 'Columns', 'Sort', and 'SQL'. The 'Condition 1' tab is active. It shows a 'Selected table(s):' dropdown with 'USERID.EMPLOYEE' selected. The 'Operator:' dropdown has 'is greater than (>)' selected. The 'Values:' field contains ':SALARY'. The 'Columns:' list on the left includes 'JOB', 'EDLEVEL', 'SEX', 'BIRTHDATE', 'SALARY', and 'BONUS', with 'SALARY' selected. Below the columns list, there is a checkbox for 'Distinct type' and a text box that says 'In table 'USERID.EMPLOYEE', find all rows in column 'SALARY' that are greater than (>) 'SALARY''. At the bottom, there are buttons for '< Back', 'Next >', 'Finish', and 'Cancel'. The right screenshot is the 'Inserting Java Stored Procedure - USERID.HighSalary' dialog box. It has tabs for '1. Name', '2. Pattern', '3. SQL Query', '4. Parameters', and '5. Options'. The '3. SQL Query' tab is active. It shows the SQL query for the stored procedure. The query is: 

```
SELECT
    EMPLOYEE.EMPNO,
    EMPLOYEE.LASTNAME,
    EMPLOYEE.SALARY,
    DEPARTMENT.DEPTNAME
FROM
    DEPARTMENT,
    EMPLOYEE
WHERE
    (
        (
            EMPLOYEE.WORKDEPT = DEPARTMENT.DEPTNO
        )
        AND
        (
            USERID.EMPLOYEE.SALARY > :SALARY
        )
    )
```

 The text 'USERID.EMPLOYEE.SALARY' is circled in red. At the bottom, there is a 'Next >' button.

you can edit the generated code before deploying to DB2

VA Java V3 - DB2 Stored Procedure Builder

© 1999 IBM Corporation

99SWB402UW

Figure 11-14. Conditions and Generated SQL

# Define Parameters and Options



## Parameters (IN, OUT, InOut)

- Name
- Java type
- SQL type

### Step 4 of 5: Parameters

For any stored procedure parameters, specify the SQL data types. (This is an optional step.)

#### Parameters

Mode	Name	Java type	SQL type	
in	SALARY	String	varchar(254)	<input type="button" value="Add"/> <input type="button" value="Change..."/>

## Options: SQLJ or JDBC

- name and package

**Step 5 of 5: Options**

Specify options for generating and creating the stored procedure.

Specific name:

Java package:

Database access:

☐ Static SQL using SQLJ

☒ Dynamic SQL using JDBC

Completion:

☒ Generate and build ☐ Generate only

☐ Build the stored procedure for debugging

**Define Parameter**

Parameter mode:

☒ In ☐ Out ☐ InOut

Name:

SQL type:

Length:

Unit:

Precision:

Scale:

☐ For bit data

VA Java V3 - DB2 Stored Procedure Builder

© 1999 IBM Corporation

99SWB402UW

Figure 11-15. Define Parameters and Options

Generate and build compiles the Java code and creates the stored procedure in DB2:

```
HIGHSALARY - Javac completed.
HIGHSALARY - Jar file created.
HIGHSALARY - sqlj.remove_jar completed.
HIGHSALARY - sqlj.install_jar completed.
HIGHSALARY - Source updated.
HIGHSALARY - Create stored procedure completed.
HIGHSALARY - Build successful.
```

## Generated Code



### Declaration of class with one public static method for the Stored Procedure

- method arguments as specified in the Stored Procedure SmartGuide
- Connection setup for JDBC
- None for SQLJ (uses defaultContext)
- Additional setup
  - ResultSet for JDBC
  - Iterators for SQLJ
- Comments with sample code to access the data in the ResultSet
  - If multiple queries were specified in the SmartGuide
  - One in parameters is an int, for indication of which query to execute
  - a switch block for selecting the query to execute
  - Sample code contains a switch block for getting access to the ResultSet
- cursor closing if the ResultSet isn't returned to the caller
- setting out parameters (but not using the set method)

VA Java V3 - DB2 Stored Procedure Builder

© 1999 IBM Corporation

99SWB402UW

Figure 11-16. Generated Code

#### Generated Java class:

```
/** JDBC Stored Procedure USERID.HIGHSALARY */
package itso.vaj3.spb;
import java.sql.*;           // JDBC classes
public class HighSalary {
    public static void highSalary ( int SALARY,
                                   ResultSet[] rs ) throws SQLException, Exception
    { // Get connection to the database
        Connection con = DriverManager.getConnection("jdbc:default:connection");
        PreparedStatement stmt = null;
        String sql;
        sql = "SELECT"
            + "      EMPLOYEE.EMPNO, EMPLOYEE.LASTNAME, EMPLOYEE.SALARY, DEPARTMENT.DEPTNAME"
            + " FROM DEPARTMENT, EMPLOYEE"
            + " WHERE"
            + "      ( ( EMPLOYEE.WORKDEPT = DEPARTMENT.DEPTNO )"
            + "      AND ( ( EMPLOYEE.SALARY > ? ) ) )";
        stmt = con.prepareStatement( sql );
        stmt.setInt( 1, SALARY );
        rs[0] = stmt.executeQuery();
        if (con != null) con.close();
    }
}
```



## Modifying Existing Code



- **Modify properties of the Stored Procedure**
  - stored Procedure
    - Name and comment
  - Parameters
    - add, delete, modify, change order
  - Options (build)
    - jar file, specific name, CREATE parameters
- **Insert SQL**
  - Opens SQL Assistant
    - Depending on the initial selected type of SQL statements (JDBC or SQLJ) SQL Assistant will insert SQL Statements of the same type
- **Edit code**
  - Edit the code as within any other IDE
  - The editor is a Java version of LPEX
  - Change to another edition of the Stored Procedure (only JDBC)

*Figure 11-17. Modifying Existing Code*

## Run a Stored Procedure



### Select run

- insert values for the parameters
- click OK

**Specify Variable Values - USERID.HIGHSALARY**

Specify the variable values to use:

☒ Treat empty strings as nulls

Name	Type	Value
SALARY	int	30000

OK Cancel Help

### Follow execution in the Messages and Results area

Returned the resultset from the stored procedure: HIGHSALARY

EMPNO	LASTNAME	SALARY	DEPTNAME
000010	HAAS	54350.00	SPIFFY COMPUTER SERVICE DIV.
000110	LUCCHESSI	47000.00	SPIFFY COMPUTER SERVICE DIV.
000020	THOMPSON	41750.00	PLANNING
000030	KWAN	38750.00	INFORMATION CENTER
000060	STERN	32250.00	MANUFACTURING SYSTEMS
000070	PULASKI	36670.00	ADMINISTRATION SYSTEMS
000050	GEYER	40175.00	SUPPORT SERVICES

7 record(s) selected.

Result Messages

Figure 11-18. Run a Stored Procedure

## Integration with VA Java



- Stored Procedure Builder from VA Java is a special version of Stored Procedure Builder
- Java classes are imported into VA Java
  - SQLJ as resource files and translated SQLJ classes
  - JDBC classes as any other Java classes
- JDBC classes are imported as open editions and are not versioned as part of the import
  - Packages are created as necessary apart from the default package
  - Package must not be version
- SQLJ classes (files) are imported as plain resource files (no versioning) and the translated classes are imported as open editions
- Versions of JDBC classes can be retrieved from Stored Procedure Builder

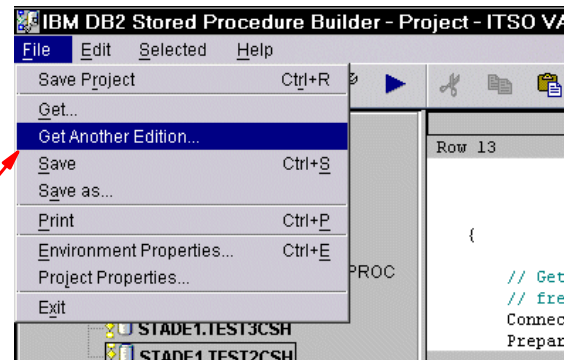


Figure 11-19. Integration with VA Java

## Using Stored Procedures



- **Call stored procedure from JDBC program**

- setup JDBC statement
- prepare statement, register variables
- execute

- **Call stored procedure from SQLJ program**

- #sql call

- **Visual Construction**

- Data Access Beans provide ProcedureCall bean

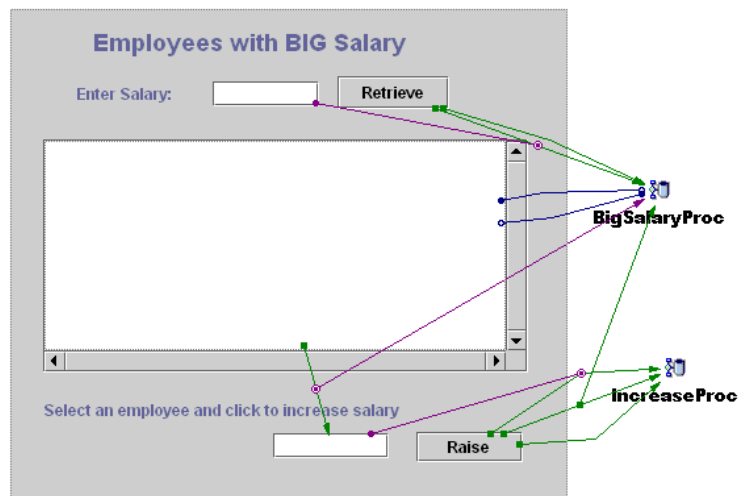


Figure 11-20. Using Stored Procedures

# Summary



## Stored Procedures

- We have seen the changes to the support for Stored Procedures introduced with DB2 V. 6.1

## Stored Procedure Builder

- We have seen the features of Stored Procedure Builder and been introduced to some limitations
- The requirements has been described

## Using Stored Procedure Builder

- We have seen how to setup and configure Stored Procedure Builder
- The process of creating a Stored Procedure, the possible choices and restrictions has been shown

## Integration with VA Java

- The integration with VA Java has been described, including some limitations

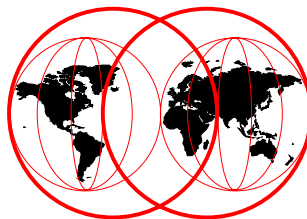
*Figure 11-21. Summary*



# 12 VA Java Version 3 SQLJ Support

**VisualAge for Java Version 3**

SQLJ Support



© 1999 IBM Corporation

99SWB402UW

## Objectives



### Quick introduction to SQLJ:

- What is SQLJ?
- Using SQLJ
- Database supporting SQLJ

### SQLJ Support in VA Java

- Features and limitations
- Using the SQLJ support in VA Java

*Figure 12-2. Objectives*

SQLJ gives native access from Java to relational databases.



## What is SQLJ?



### **SQLJ is a standard that enables the use of static SQL in Java applications**

- SQLJ is an ANSI Standard, initially developed jointly by Oracle, Tandem and IBM
- The use of static SQL increases performance as the setup of the query (authority, syntax, semantics and logic checking) is done at compile time and not at run time
- Static SQL has no features for dynamic queries, all SQL queries are completely defined at compile time - that is all SQL statements, table and column names have to be known and defined (hardcoded) at compile time
- SQLJ and JDBC can be used in the same application giving the developer the best of two worlds
- Security: GRANT authorization to users based on package

**SQLJ - heavily promoted by IBM and Oracle**

*Figure 12-3. What is SQLJ?*

## SQLJ Specification



### The SQLJ Specification consists of 3 parts:

- Part 0 specifies the SQLJ language syntax and semantics for embedded SQL statements in a Java application
- Part 1 specifies extensions that define:
  - Installation of Java classes in an SQL database
  - Invocation of static methods as stored procedures
- Part 2 specifies extensions for accessing Java classes as SQL distinct types

Most database vendors have implemented part 0

We focus only on part 0 within VA Java

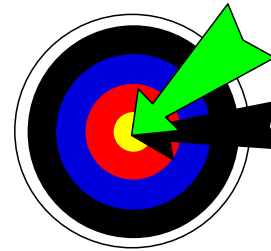


Figure 12-4. SQLJ Specification

## Using SQLJ (1)



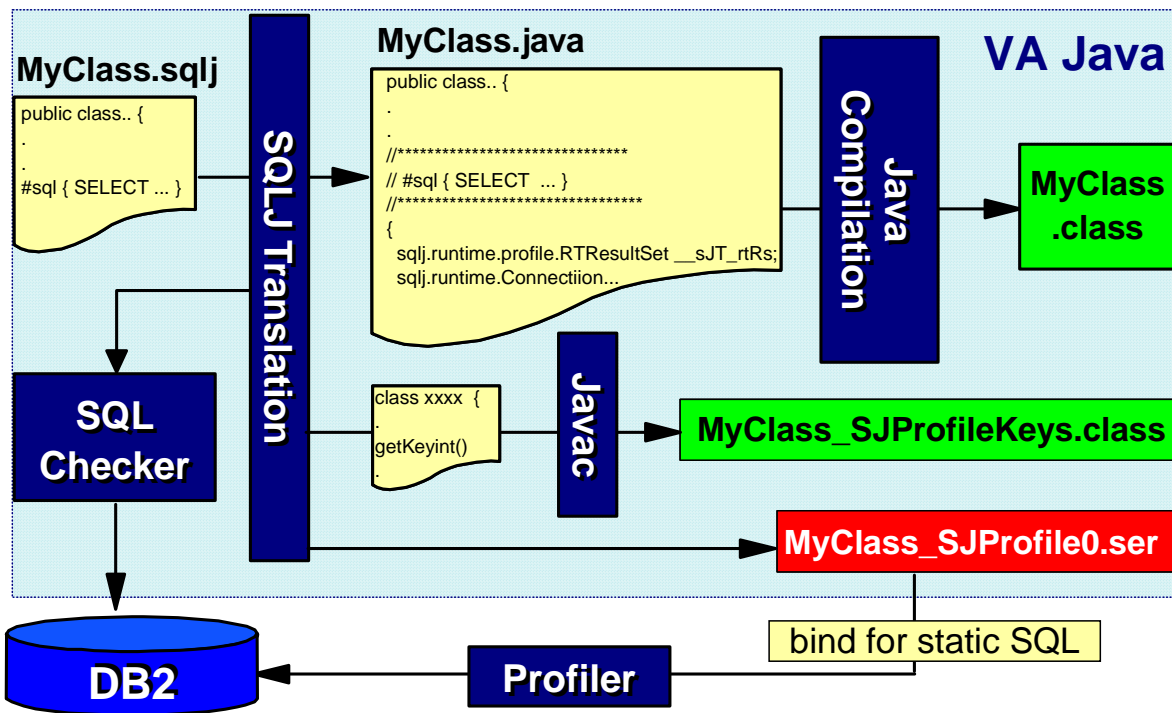
### Using SQLJ adds 2 or 3 steps to the normal Java development process:

- First one has to create the SQLJ source code file. That is a file with Java code and embedded SQL statements that are enclosed by `#sql` and `;`;
  - The SQLJ naming convention requires all Java source files with embedded SQL to have an extension of `*.sqlj` (case sensitive)
- The Java compiler cannot handle the embedded SQLJ statements, therefore they have to be *translated* to pure Java code (into a `*.java` file) by the **SQLJ Translator**
- The translated SQLJ code can be compiled with the rest of the Java application
- To gain the most (performance wise) from the SQLJ code, it can be *customized* using a tool called the **Profiler**

*Figure 12-5. Using SQLJ*

The Application Development Guide, coming with DB2 UDB version 6.1 has a very good introduction to SQLJ programming. Page 545 until page 557. It is recommended reading.

## Using SQLJ (2)



VA Java V3 - SQLJ Support

© 1999 IBM Corporation

99SWB402UW

Figure 12-6. Using SQLJ (2)

The source file (.sqlj) is run through the SQLJ Translator, and a Java source file is produced.

The Java source file is compiled as usual into a class file.

The SQL Translator also produces a serialized profile (.ser) and a profile key class.

The database specific Profiler is used to bind the SQL into the database system. The Profiler runs outside of VA Java.

## Databases Supporting SQLJ



- All databases can support SQLJ based applications without customization, provided the databases support the JDBC interface and the SQLJ used is based on the reference implementation (which uses JDBC as the default interface to the database).
- Customization requires explicit support from the database product
- DB2 on all IBM platforms support the JDBC interface and will therefore also support SQLJ applications without customization
- Currently the following DB2 platforms have support for customization of SQLJ code
  - DB2 UDB version 5.2 (Unix, OS/2, Windows NT, 95 and 98 ), Fixpak 8 is highly recommended
  - DB2 UDB version 6.1 (Unix, OS/2, Windows NT, 95 and 98 )
  - DB2 UDB server for OS/390 Version 6.0
- SQLJ translator/profile are in the DB2 Software Developer's Kit

*Figure 12-7. Databases Supporting SQLJ*

The following vendors officially supports SQLJ:

Oracle, IBM, Sybase, Informix, Compaq, CloudScape

## Translating the SQLJ File



### Command: `sqlj file.sqlj`

- The translation process replaces (comments out) the embedded SQL code with the proper Java classes used for the implementation of SQLJ
- The output from the translation process consists of a file (\*.java) with the translated SQLJ code, a profile file (\*.ser) containing all the database related information, a class file with Profile key information (\*\_SJProfileKey.class) and eventual Java class files for holding result sets from the SQL queries (cursors)
- The translated SQLJ file is compiled by the Java compiler into class files
- Never-ever change the code in the translated sqlj file, the only place to change SQLJ code is in the original sqlj file

```
sqlj -user=userid/password -url=jdbc:db2.sample  
-status -profile=false -ser2class SqljServBean.sqlj
```

*Figure 12-8. Translating the SQLJ File*

## Customizing the Profile



- The profile file (\*.ser) can be used as is (queries are done using the JDBC interface) or can be customized to a specific database (binding to the database) thereby increasing the performance
- The customizing process generates bind files and bind them to the database
- Customizing changes the profile file to make it use the packages created in the database during the bind process
- The customizing tool (the profiler) for DB2 is called **db2prof.c.exe**

```
db2prof.c [-user=userId -password=xxx -preoptions="..."]  
          -url=jdbc:db2:sample  
          <profilename>.ser
```

*Figure 12-9. Customizing the Profile*

The real performance advantage of SQLJ comes from the profile.

The database system provides a Profiles that binds the profile into the database systems, similar to binding the file produced when compiling 3GL program with embedded SQL.

## Advantages of SQLJ



### The advantages of SQLJ - compared to JDBC - include:

- Faster execution at runtime
- Enables binding, allowing access control based on access to packages
- Simpler code
  - Calling stored procedures is very simple using SQLJ. When using JDBC to call a stored procedure, the setup of the variables and retrieval of the output is the source of numerous statements.
  - Using SQLJ, 30 lines of JDBC code used to call a stored procedure can be replaced by 5 - 7 statements when using SQLJ.
  - The difference becomes more obvious, the higher the number of variables used for calling the stored procedure.

*Figure 12-10. Advantages of SQLJ*



# Calling Stored Procedures using JDBC



## DB2SPCli.java:

```
public static void callStoredProc (Connection con,
    String tableName,
    String name,
    int percentModification,
    String department) throws Exception
{ // prepare the CALL statement
    CallableStatement stmt;
    String sql = "Call " + name + "(?, ?, ?, ?, ?, ?, ?) ";
    stmt = con.prepareCall (sql);

    // register the output parameters
    stmt.registerOutParameter (4, Types.DOUBLE);
    stmt.registerOutParameter (5, Types.DOUBLE);
    stmt.registerOutParameter (6, Types.DOUBLE);
    stmt.registerOutParameter (7, Types.DOUBLE);
    stmt.registerOutParameter (8, Types.INTEGER);
    stmt.registerOutParameter (9, Types.CHAR);

    // set all parameters (input and output)
    double totalPayrollBefore = 0.00;
    double totalPayrollAfter = 0.00;
    double averageSalaryBefore = 0.00;
    double averageSalaryAfter = 0.00;
    String medianEmployeeName = "This field is not defined yet";
    int numberOfUpdates = 0;

    stmt.setString (1, tableName);
    stmt.setInt (2, percentModification);
    stmt.setString (3, department);
    stmt.setDouble (4, totalPayrollBefore);
    stmt.setDouble (5, totalPayrollAfter);
    stmt.setDouble (6, averageSalaryBefore);
    stmt.setDouble (7, averageSalaryAfter);
    stmt.setInt (8, numberOfUpdates);
    stmt.setString (9, medianEmployeeName);

    // call the stored procedure
    System.out.println ("\n Calling stored procedure: " + name);
    stmt.execute ();
    System.out.println ("\n Returned from stored procedure: " + name);

    // retrieve output parameters
    BigDecimal totPayBefore = new BigDecimal (stmt.getDouble (4));
    BigDecimal totPayAfter = new BigDecimal (stmt.getDouble (5));
    BigDecimal avgSalBefore = new BigDecimal (stmt.getDouble (6));
    BigDecimal avgSalAfter = new BigDecimal (stmt.getDouble (7));
    numberOfUpdates = stmt.getInt (8);
    medianEmployeeName = stmt.getString (9);
    .....
    // the procedure continues with printing the result
```

Figure 12-11. Calling Stored Procedures using JDBC

## Calling Stored Procedures Using SQLJ



```
public static void callStoredProc (Connection con,
                                   String tableName,
                                   String name,
                                   int percentModification,
                                   String department) throws Exception
{
    // create execution context (defaultcontext), con is a JDBC connection
    DefaultContext ctx = new DefaultContext();
    DefaultContext.setDefaultContext(ctx);

    // set all parameters not already set(input and output)
    double totalPayrollBefore = 0.00;
    double totalPayrollAfter = 0.00;
    double averageSalaryBefore = 0.00;
    double averageSalaryAfter = 0.00;
    String medianEmployeeName = "This field is not defined yet";
    int numberOfUpdates = 0;

    System.out.println ("\n Calling stored procedure: " + name);
    #sql {CALL :name(:IN tableName, :IN percentModification,
                  :IN department, :OUT totalPayrollBefore, :OUT totalPayrollAfter,
                  :OUT averageSalaryBefore, :OUT averageSalaryAfter,
                  :OUT numberOfUpdates, :OUT medianEmployeeName)
    };
    System.out.println ("\n Returned from stored procedure: " + name);
    .....
    // the procedure continues with printing the result
}
```

VA Java V3 - SQLJ Support

© 1999 IBM Corporation

99SWB402UW

*Figure 12-12. Calling Stored Procedures Using SQLJ*

One could argue that this code isn't equivalent to the JDBC code, because it lacks the instantiation of the `BigDecimal` types. But the point is that each of these instantiations contains a `getDouble` statement which is superfluous using SQLJ. So to show the point, the difference in number of statements, these instantiations had to be removed.

The code presented here has been tested and will indeed call the stored procedure as expected.

## VisualAge for Java Support for SQLJ



### Features and limitations:

- **What VA Java does support:**

- Importing \*.sqlj files
- Editing \*.sqlj files
- Translation of the \*.sqlj file
- Debugging using the \*.sqlj file as reference

- **What VA Java doesn't support:**

- Creation of \*.sqlj files
- Version control of SQLJ files (you will need another tool for that)
- Customizing the profile file (calling the Profiler)

**Make sure that the .sqlj file is read-write (read-only files fail in translation)**

*Figure 12-13. VisualAge for Java Support for SQLJ*

## Setting up VA Java SQLJ Support



### Adding the SQLJ feature:

- From the workspace click **File->Quickstart**
- select the SQLJ Runtime Library

### Configuring the SQLJ environment:

- Workspace -> Tools -> SQLJ -> Properties
- Two main features can be configured:
  - The **encoding** used during the translation. This is used for translating non-ASCII characters in the embedded SQL statements. If no encoding is specified, default is ISO8859\_1 (Latin-1).
  - **Online Semantic Checking** for checking the table and field names used in the embedded SQL statements
    - The database driver must be included in the class path of the translator (db2java.zip file in **Window->Options->Resources**)
    - COM.ibm.db2.jdbc.app.DB2Driver, and jdbc:db2:databasename

*Figure 12-14. Setting up VA Java SQLJ Support*

A one-time setup is required before using SQLJ.

## Using VA Java to Create the SQLJ file



### Use VA Java to create the SQLJ file (workaround):

- In VA Java, create a class of the name you want to use for the SQLJ file (this way all the javadoc information you have setup for a type is included in the SQLJ file)
- Export it to the filesystem (export to directory) -> a file - *classname.java* - is created
- Rename it to *classname.sqlj*
- Import the sqlj file into VA Java. Select you project, right hand click it and select **Tools->SQLJ->Import**. Deselect Perform translation in the import window. Do not use the regular import facility!
- Verify that the file has been imported by going to the Project Browser and select the Resources Page. Here the sqlj files you have imported are listed (press **F5** to have the window refreshed).
- Remove the copy of the file in the file system to be sure that you are not working on the same file in 2 places

*Figure 12-15. Using VA Java to Create the SQLJ file*

VA Java does not provide an editor or SmartGuide to create an SQLJ file.

You can, however, create a Java class with some initial code, export it, rename it to .sqlj, and then work with it through the Resource Browser in the project.

## Editing the SQLJ File



- Select the sqlj file and right hand click it and select **Tools->SQLJ->Edit** from the menu
  - The editor used is a Java version of IBM's Lpex editor
- To use another editor for the SQLJ files:
  - Select **Window->Options**
  - Under **Resource->Resource Associations** it is possible to setup up associations for resource extensions
  - To use the alternative editor, double-click the sqlj file and it will open into the editor configured
- VA Java does not support any versioning of the SQLJ file!

*Figure 12-16. Editing the SQLJ File*

The editor used for editing SQLJ files are a java version of LPEX.

Currently there is no documentation for this editor in VAJ help.

On the intranet documentation of the editor can be found at  
<http://ewb1.torolab.ibm.com/javalpex/index.html>

When using javalpex for the first time, one notes that the Tab key doesn't work as expected (not moving text only moving the cursor. The default action for pressing the Tab key is 'nextTabStop'.

By entering 'set keyAction.tab insertTab' on the command line of the editor, pressing the Tab key will insert a tab character which will be displayed as 8 characters regardless of tab stops, etc.

In a later version this maybe changed to allow the insertion of spaces to the next tab stop instead.

## Commands for the SQLJ Editor (1)



### Text editing commands:

Ctrl+C	Copies selected text to the clipboard
Ctrl+V	Pastes text from the clipboard
Ctrl+X	Cuts selected text from the editor and places it in the clipboard
Ctrl+Backspace	Deletes the line containing the cursor
Ctrl+Enter	Inserts a new line beneath the cursor
Ctrl+Shift+End	Selects all text from the cursor downwards
Ctrl+Shift+Home	Selects all text from the cursor upwards
Ctrl+Shift+Left Arrow	Selects the previous word (or selects a portion of the current word)
Ctrl+Shift+Right Arrow	Selects the next word (or selects a portion of the current word)
Alt+L	Selects line of text containing the cursor; move cursor and press Alt+L again to select a group of lines

*Figure 12-17. Commands for the SQLJ Editor (1)*

## Commands for the SQLJ Editor (2)



### Text editing commands (2):

Alt+B	Selects a block of text starting with cursor; move cursor and press Alt+B again to mark the end of the block
Alt+R	Selects a rectangular block of text starting with the cursor; move cursor and press Alt+R again to mark the end of the block
Alt+U	Deselects selected text
Ctrl+Z	Undoes changes; press Ctrl+Z repeatedly to undo multiple changes
Ctrl+Shift+Z	Redoes changes; press Ctrl+Shift+Z repeatedly to redo multiple changes

*Figure 12-18. Commands for the SQLJ Editor (2)*



## Commands for the SQLJ Editor (3)



### Table of search commands:

Ctrl+F	Finds specified text
Ctrl+J	Finds the last change made
Ctrl+M	Finds the matching bracket
Ctrl+N	Finds the next occurrence of specified text (searching downwards)
Ctrl+U	Finds the previous occurrence of specified text (searching upwards)
Ctrl+A	Displays all lines in file

### Table of movement commands:

Ctrl+Home	Moves the cursor to the top of the file
Ctrl+End	Moves the cursor to the end of the file
Ctrl+Left Arrow	Moves the cursor to the beginning of the previous word
Ctrl+Right Arrow	Moves the cursor to the beginning of the next word
Ctrl+L	Moves the cursor to the specified line number

*Figure 12-19. Commands for the SQLJ Editor (3)*

## Translating the SQLJ File



- Select the sqlj file, right hand click it and select **Tools->SQLJ->Translate** from the menu
- The translation ends with a message saying the translation was completed, no matter what the result of the translation was
- During the translation VA Java creates a new edition of the translated SQLJ class. You will have to version the translated class yourself.
- The profile (.ser) file is in ide\project\_resources\  
your-project\package\

**– the Profiler (db2profc) must be run outside of VA Java**

*Figure 12-20. Translating the SQLJ File*

You can translate an SQLJ file from within the VA Java IDE.

The profiler must be run outside against the database system.

## SQLJ Translator Output (1)



### Understanding the translator output:

- The translator does a simple syntactical and semantic analysis of the Java code, before commencing the actual translation
- It will output 3 different types of messages:
  - **Info** is to inform the user that a major part of the translation has been undertaken and - if no errors or warnings are displayed - has been completed successfully
  - **Warning** is used whenever the result of a part of the translation process is negative but the translation of the sqlj file can continue
  - **Error** is used when something stops the translator from translating or parsing the whole SQLJ file . The reason for an error message is normally a major syntactical error (like a missing semicolon or unmatched curly braces).

Figure 12-21. SQLJ Translator Output (1)

## SQLJ Translator Output (2)



### Understanding the translator output:

Typical output from the translator looks like this (displaying an error):

```
[Translating 1 files.]  
[Reading file SqljBean]  
C:\IBMJava\ide\Program\...\ivjtools\temp\Temp6813\SqljBean.sqlj:50.8:  
Error: Missing semicolon.  
  
Total 1 error.
```

Any error, warning or information in the output contains a line and column number reference to the part of the code that caused the error

When the the information relates to a `#sql` statement, it includes the starting and ending reference of the statement.

Whenever an error occurs in a SQL statement, the translator relies on the output from DB2 and the JDBC driver to explain the error.

Figure 12-22. SQLJ Translator Output (2)

## SQLJ Translator Output (3)



### Understanding the translator output:

- Output from a translation doing online semantic checking

```
[Translating 1 files.]
[Reading file SqljBean]
[Translating file SqljBean]
C:\IBMJava\ide\Program\...\ivjtools\temp\Temp3213\SqljBean.sqlj:69.3-69.92:
Info: [Registered JDBC drivers: COM.ibm.db2.jdbc.app.DB2Driver]
C:\IBMJava\ide\Program\...\ivjtools\temp\Temp3213\SqljBean.sqlj:69.3-69.92:
Info: [Connecting to user stadel at jdbc:db2:SAMPLE]
C:\IBMJava\ide\Program\...\ivjtools\temp\Temp3213\SqljBean.sqlj:69.3-69.92:
Info: [Querying database with "SELECT empno, lastname, workdept, sex FROM
employee where ( workdept = ? ) "]
C:\IBMJava\ide\Program\...\ivjtools\temp\Temp3213\SqljBean.sqlj:72.3-72.73:
Info: [Querying database with "SELECT COUNT(*) FROM employe1 where (
workdept = ? ) "]
C:\IBMJava\ide\Program\...\ivjtools\temp\Temp3213\SqljBean.sqlj:72.3-72.73:
Warning: Unable to check SQL statement. Error returned by database is:
[IBM][CLI Driver][DB2/NT] SQL0204N "STADE1.EMPLOYE1" is an undefined name.
SQLSTATE=42704
Total 1 warning.
```

Figure 12-23. SQLJ Translator Output (3)

## Debugging SQLJ Code



### Debugging SQLJ code:

- Whenever running or debugging SQLJ code using the IDE, remember to add the package runtime.zip to the resource classpath using **Window->Options->Resources**, if not already done. Otherwise the SQLJ classes cannot run. If you are using DB2, runtime.zip can be found in `d:\sqllib\java`.
  - some versions of DB2 also have runtime.zip and sqlj.zip
- When debugging in the IDE, the debugger will reference the translated Java file not the original sqlj file.
- One can export a classfile that contains debug information referring to line numbers of the original sqlj file. This class can be used with a stand-alone debugger.

*Figure 12-24. Debugging SQLJ Code*

## Summary



- **Quick introduction to SQLJ**

- We have seen where SQLJ comes from, what it does and how it works
- We have seen one of the advantages of SQLJ, the code gets simpler

- **SQLJ support in VA Java**

- We have seen the support for SQLJ that VA Java provides and the limitations of this support
- We described how to use this support

*Figure 12-25. Summary*

SQLJ is the high-performance option for Java access to relational databases and VisualAge for Java has good support for SQLJ in its IDE.

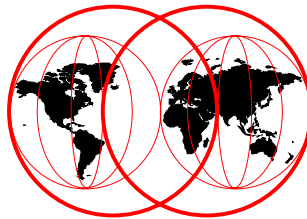




# **13 VA Java Version 3 Tool Integration API**

## **VisualAge for Java Version 3**

**Remote Access to Tool API**  
(Remote Access Server)



© 1999 IBM Corporation

99SWB402UW

## Objectives



### **Understand the new tool integration interface of VA Java V3**

- Remote Access to Tool API
  - Remote Access Server (Tool Server)
  - Tool Servlet
- How to start Remote Access Server
- How to develop/debug a Tool Servlet

**You will also see why this function is added to VA Java**

*Figure 13-2. Objectives*

VisualAge for Java Version 3 provides a better API for tool builders.

In Version 2 access to repository/workspace data was limited to tools running inside the IDE.

Version 3 provides a remote API for tool builders.

## Tool API



### **Tool API provides the way to access VA Java functions and manipulate program elements**

- Browse workspace and repository
- Query objects (projects/classes/interfaces)
- Invoke VA Java functions
  - Import/Export from/to source files, .class files and repository files
  - Load editions from the repository into the workspace
  - Open a browser / SmartGuide
- Work with class/interface source code
- Access tool data

### **Tool API is ideally suited to tool vendors and ISVs**

- You can create a tool integrated to VA Java IDE

### **Tool API is available only inside VA Java IDE**

- From outside VA Java IDE, you cannot call Tool APIs
- You cannot integrate tools running in another process/machine easily

*Figure 13-3. Tool API*

The Tool API was introduced in Version 2. It provides access to workspace and repository data.

This API has been used by many of the tools that are integrated into the VA Java IDE.

## Remote Access to Tool API

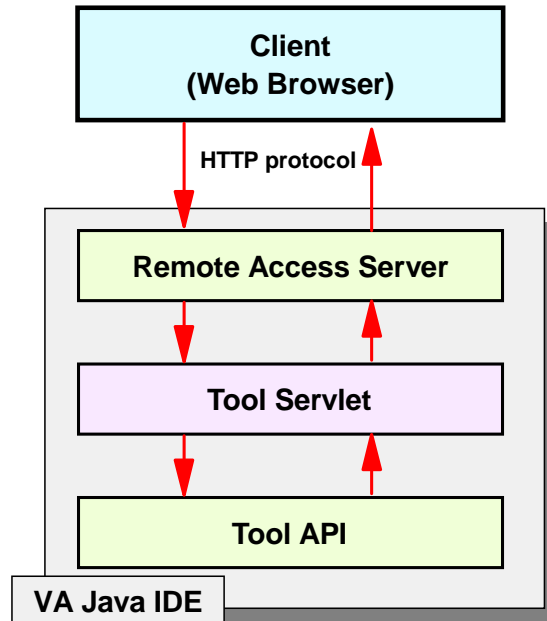


**Provides the way to access VA Java functions and manipulate program elements**

- from **outside** VA Java IDE
- through HTTP protocol

**Consists of:**

- **Remote Access Server (Tool Server)**
  - is a lightweight HTTP daemon
  - is running inside VA Java IDE
  - supports Tool Servlets
- **Tool Servlet**
  - accepts a request from a client
  - performs a task using Tool API
  - returns a response to the client



*Figure 13-4. Remote Access to Tool API*

New in Version 3 is the remote access to the Tool API.

A remote access server inside VA Java provides access to outside tools. The tools provides a servlet (tools servlet) inside VA Java for data access through the regular Tool API.

## Starting Remote Access Server



### Use Options Dialog to start or stop Remote Access Server

- You can
  - start it automatically at VA Java IDE startup
  - view its trace log in **Console** Window
  - specify a port number
  - specify hosts permitted to access
    - default: localhost only
- A port number is stored to:
  - `toolport.properties(*)`
- You can get system-generated port number from this file
- Access permitted hosts are stored to:
  - `access.properties(*)`

**Remote Access To Tool API**

☐ Start Remote Access To Tool API on VisualAge startup

☐ Enable tracing of Remote Access To Tool API

☒ Use system-generated port

☐ Use user-defined port

Remote Access To Tool API is not currently running

Access permitted from the following hosts:

(\*) in `\IBMJava\ivjtools\tooldata\com-ibm-ivj-toolserver` directory

Figure 13-5. Starting Remote Access Server

To activate the remote tools API the Remote Access Server must be started.

## Tool Servlet



**To create a Tool Servlet, you should be familiar with:**

- Tool API
- Servlet and Java Servlet API (V2.0)
  - Developing Tool Servlets is almost the same as developing standard servlets, but ....

**You *must* use Tool Servlet packages instead of standard servlet packages**

- javax.servlet -> com.ibm.ivj.toolserver.servletclasses.servlet
- javax.servlet.http -> com.ibm.ivj.toolserver.servletclasses.servlet.http
- sun.servlet -> com.ibm.ivj.toolserver.server.servlet
- sun.servlet.http -> com.ibm.ivj.toolserver.server.servlet.http

**You must deploy .class files of Tool Servlets to:**

- \IBMJava\ide\TOOLS\com-ibm-ivj-toolserver\servlets directory

*Figure 13-6. Tool Servlet*

A tool servlet is almost like a regular servlet. It must use a special version of the servlet API classes that is provided by VA Java.

## Developing Tool Servlet



### To Develop a Tool Servlet in VA Java you have to:

- Add the **IBM IDE Utility Class Libraries** feature
- Import Tool Servlet packages as .class files from:
  - \IBMVJava\ide\TOOLS\com-ibm-ivj-toolserver\com
- Create your Tool Servlet inherited from:
  - com.ibm.ivj.toolserver.servletclasses.servlet.http.HttpServlet
  - Make sure you are using Tool Servlet packages instead of standard servlet packages

### Test and debug your Tool Servlet using VA Java debugger

– see next page

### Deploy your Tool Servlet to:

- \IBMVJava\ide\TOOLS\com-ibm-ivj-toolserver\servlets

### Access your Tool Servlet using Web Browser etc...

- URL is `http://localhost:port-num/servlet/package.class`

Figure 13-7. Developing Tool Servlet

## Debugging Tool Servlet



**To debug your Tool Server using VA Java debugger, you must start Remote Access Server as a program**

- Stop Remote Access Server, if it is running
- Run `com.ibm.ivj.toolserver.ToolHttpServer` class from IDE
  - Before you run it, add the following projects and external directories to the class path
    - Project: IBM IDE Utility Class Libraries
    - Project: in which your Tool Servlet is contained
    - Directory: \IBMJava\ide\project\_resources\IBM IDE Utility local implementation
    - Directory: \IBMJava\ide\project\_resources\IBM IDE UI class libraries
  - Before you run it, set the following command line arguments
    - `-v -p port-number -d .`

**Set breakpoints in your Tool Servlet**

**Invoke your Tool Servlet from Web Browser etc..**

*Figure 13-8. Debugging Tool Servlet*



## Assigning Aliases to Tool Servlets



**Define aliases of Tool Servlets in the `servlet.properties` file. This file located in:**

- `\IBMJava\ide\TOOLS\com-ibm-ivj-toolserver\servlets`

**Add the following line to define an alias**

- `servlet.alias-name.code=yourpkg.YourToolServlet`

**Invoke the Tool Servlet using following URL**

- `http://localhost:port-num/servlet/alias-name`

*Figure 13-9. Assigning Aliases to Tool Servlets*

Tool servlets can be assigned short names (aliases) that make the calling sequence from a Web browser shorter.

## Samples



### Source files of sample Tool Servlets are provided in:

- \IBMVJava\ide\TOOLS\com-ibm-ivj-toolserver\  
    servlets\code\com\ibm\ivj\toolserver\samples
  - ExportFileServlet
  - ImportFileServlet
  - LaunchBrowserServlet

### You can invoke sample Tool Servlets from the following HTML file

- \IBMVJava\ide\TOOLS\com-ibm-ivj-toolserver\  
    servlets\CallSampleServlets.html

*Figure 13-10. Samples*

The best way to understand the remote Tool API is to look at the samples provided with VA Java.

## Summary



**Using Remote Access to Tool API, you can easily access to VA Java IDE from another process/machine through HTTP protocol**

**You can integrate tools running in another process/machine into VA Java IDE**

*Figure 13-11. Summary*

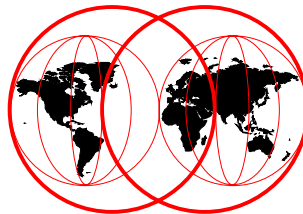
The Remote Tool API opens the world to a new set of potential VA Java tools.



# **14 VA Java Version 3 Enterprise Access Builders Transactions**

## **VisualAge for Java Version 3**

Enhanced Enterprise Access Builder for  
Transactions (EAB)



## Objectives



### **Understand the enhancements of Enterprise Access Builder for Transactions (EAB) provided by VA Java Enterprise Edition V3**

#### **New EAB provides you :**

- New SmartGuides
- A new record importer
- Enhancement of :
  - Java Record Framework and COBOL record
  - Mapper and Mapper Editor
- EAB Session Bean Tool
- New Connectors

### **Before we show you enhancements provided by VA Java V3, let's look back on base items and frameworks**

VA Java V3 - EAB Transactions

© 1999 IBM Corporation

99SWB402UW

*Figure 14-2. Objectives*

The Enterprise Access Builder has been renamed to Enterprise Access Builder for Transactions.

The SmartGuides have been improved and the flow between steps has been enhanced.

The new EAB Session tool generates EJBs for EAB transactions.

But first, let us review the functions of the Enterprise Access Builder of Version 2.

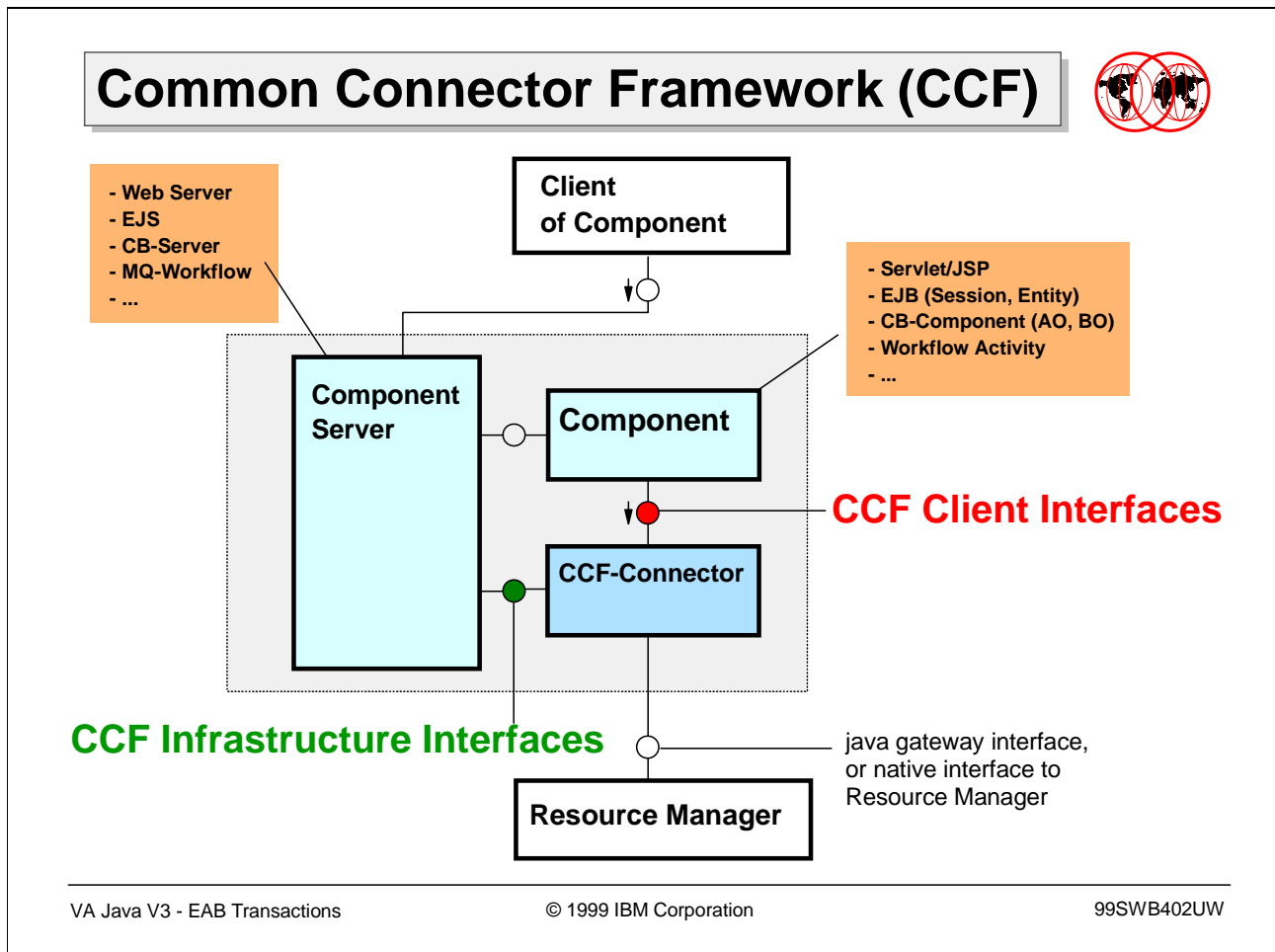


Figure 14-3. Common Connector Framework (CCF)

The Common Connector Framework was introduced with Version 2 as a base for the CICS (and Encina) Connector.

It provides a standard interface to Java applications accessing enterprise resources (for example, CICS transactions).

# Java Record Framework

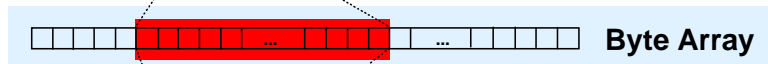


A Record wraps a data buffer that is sent to a server and received from a server, and it is based on Java Record Framework

This framework supports :

- data conversion
- nested record
- overlay
- array
  - fixed size
  - variable size
- alignment

```
public String getFirstName();  
public void setFirstName(String firstName);
```



```
01 DFHCOMMAREA.  
  02 CustNo      PIC X(5).  
  02 FirstName   PIC A(15).  
  02 LastName    PIC A(25).  
  02 Street      PIC X(20).  
  02 City        PIC A(20).  
  02 Country     PIC A(10).  
  02 Phone       PIC X(15).  
  02 PostalCode  PIC X(7).  
  
COBOL  
copybook
```

Figure 14-4. Java Record Framework

The Java Record Framework provides the base to pass records, such as a communications area, from Java applications to enterprise transaction systems.

The initial support was for COBOL structures that are translated into Java classes.



## Record Type and Record



### A Record Type is a meta data of a Record

- You can create/edit a Record Type using Record Editor and generate a record from it
- You can also create a Record Type by importing an existing record definition, such as COBOL source file

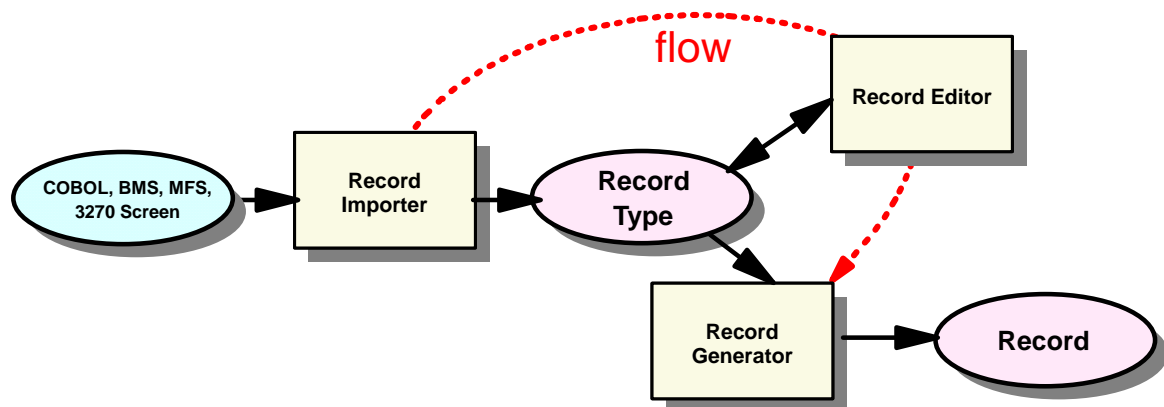


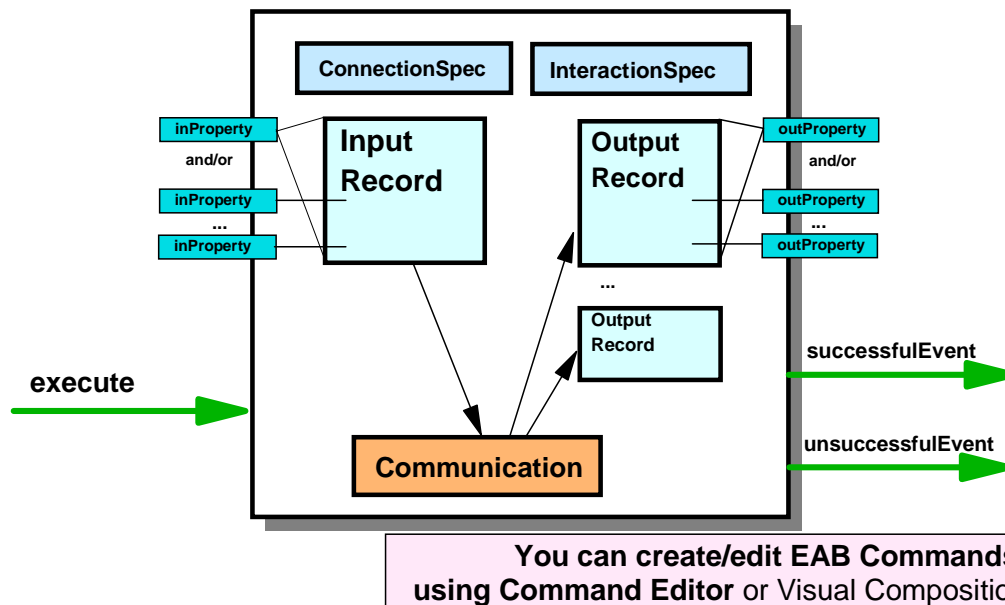
Figure 14-5. Record Type and Record

A record type can be imported from existing definitions, edited, and then a record is generated, to be passed through commands to enterprise systems.

## EAB Command



An EAB Command wraps a *single* interaction between Java program and your Enterprise Information System (EIS)



VA Java V3 - EAB Transactions

© 1999 IBM Corporation

99SWB402UW

Figure 14-6. EAB Command

The Command is the vehicle for one interaction between the Java application and the enterprise system.

The Command includes an input record and output record(s).

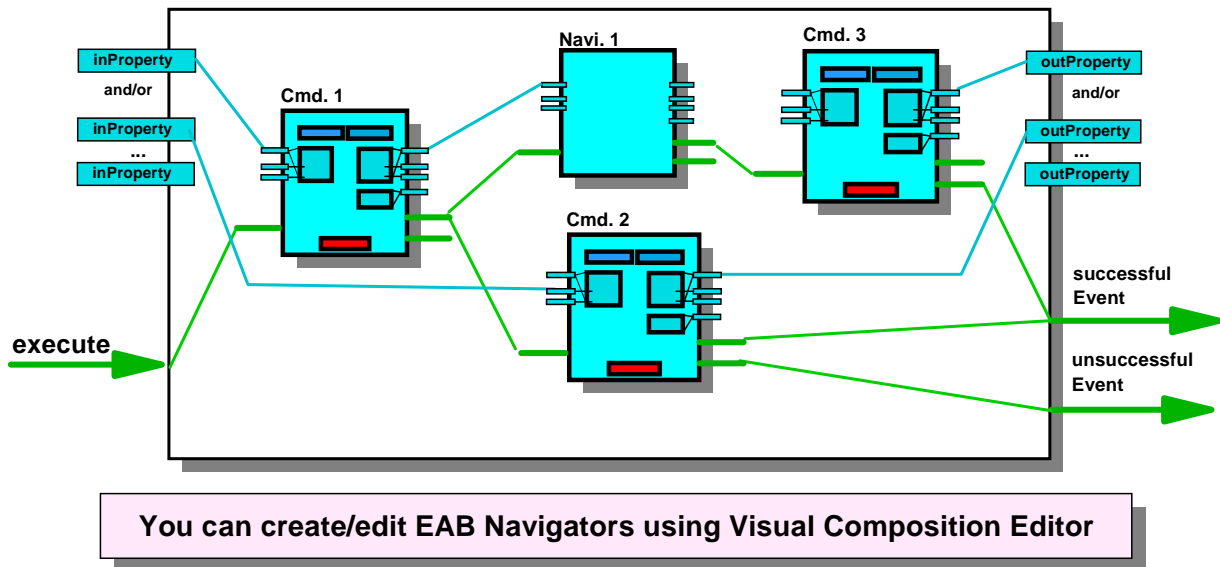
The Command must have a connection specification (which enterprise system) and an interaction specification (which program/transaction).

These specifications are tailored for the enterprise system to be accessed.

## EAB Navigator



An EAB Navigator wraps *multiple* interactions between Java program and your Enterprise Information System (EIS)



VA Java V3 - EAB Transactions

© 1999 IBM Corporation

99SWB402UW

Figure 14-7. EAB Navigator

The Navigator can be used to chain multiple commands into one operation (in the view of the Java application).

It also provides nice facilities for just one interaction (command).

## Mapper and Business Object



**A Mapper is a bean mapping properties between :**

- a bean <-> a bean
- a collection of beans <-> a bean

**In EAB, a Mapper is used to map properties between a record and application object(s) such as :**

- BMP Entity bean
- EAB Business Object
- Any kind of business object

You can create/edit Mapper using Mapper Editor

**EAB Business Object is a type of business object defined by VA Java**

- In EAB, you have the option of using EAB Business Objects
- A Mapper supports EAB Business Objects that have key in special way
  - they are stored in an Instance Space and retrieved from the Instance Space using Key

*Figure 14-8. Mapper and Business Object*

The application deals with business objects. The Command deals with records. Java code is required to copy data (attributes) from business objects to records, and vice versa.

A mapper can automate this task. Before execution of a Command, the mapper copies data from business objects to the input record, and after execution the mapper copies data from output records to business objects.

## EAB Enhancements



**Now, we show you EAB enhancements provided by VA Java Enterprise Edition V3 in the following order :**

- Enhanced Ease of Use
- Record Type Importer
- Java Record Framework and COBOL record
- EAB Command
- Mapper
- EAB Session Bean Tool
- New Connectors

*Figure 14-9. EAB Enhancements*

Now we want to look at the enhancements.

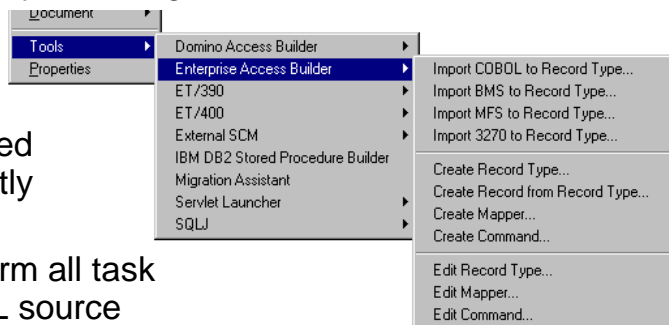
## Enhanced Ease of Use



### Consistent look and feel for all EAB tools

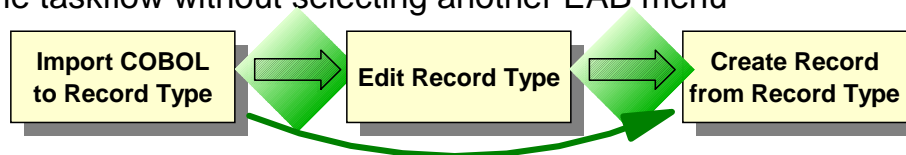
### All EAB functions in a single menu

- You can invoke all features by selecting **Tools -> Enterprise Access Builder** from popup menu



### Tool to tool taskflow

- You can invoke a tool needed in next step from the currently working tool
- For example, you can perform all task from importing a COBOL source to generating a record in one taskflow without selecting another EAB menu



VA Java V3 - EAB Transactions

© 1999 IBM Corporation

Figure 14-10. Enhanced Ease of Use

The first enhancement is not really new function, but more how the function is packaged.

In Version 3 you can invoke the next step in the record creation activities directly from the previous GUI instead of having to exit and restart with another tool.

# Record Type Importers



## New EAB provides you 4 types of Record Importer

- **Import COBOL to Record Type** Enhanced & New UI
  - You can create a record type from a COBOL source file
  - This importer was enhanced in VA Java V3
- **Import BMS to Record Type** New UI
  - You can create a record type from a CICS BMP source file
  - This importer has the same feature with old one, but new EAB provides you better user interface
- **Import MFS to Record Type** New UI
  - You can create a record type from an IMS MFS source file
  - This importer has the same feature with old one, but new EAB provides you better user interface
- **Import 3270 to Record Type** brand-new
  - This is a brand-new importer
  - You can create a record type from a 3270 screen
  - If you have no BMP/MFS source file, you can easily create a record type

*Figure 14-11. Record Type Importers*

The import functions have been enhanced with new user interfaces.

Also there is a new importer, from 3270 screens.

## Import COBOL to Record Type



### New COBOL importer supports :

- COBOL copybook
  - You have to place copybooks in the same directory with the source or in the directory specified in the source
- OCCURS DEPENDING ON
  - Variable size array
- LEVEL 88
  - Condition name  
(Possible Values)
- BigDecimal
  - You can use `java.math.BigDecimal` or `double` to access data that contain a decimal in PICTURE definition, such as 999.99
  - In VA Java V2, you always have to use `double`

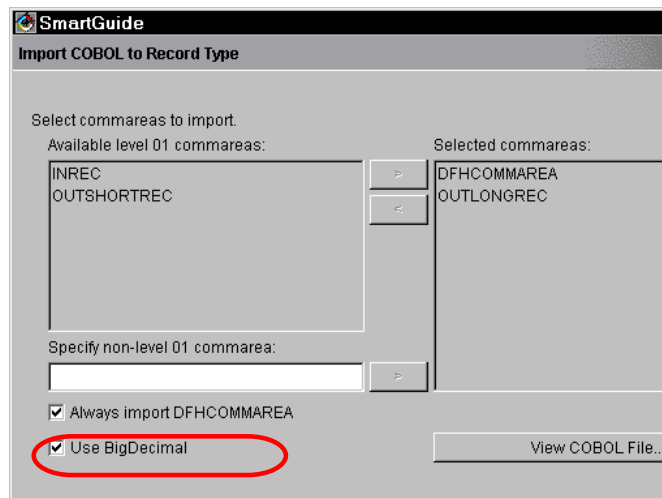


Figure 14-12. Import COBOL to Record Type



## Import 3270 to Record Type (1)



This importer is very useful when you have no BMP/MFS source files

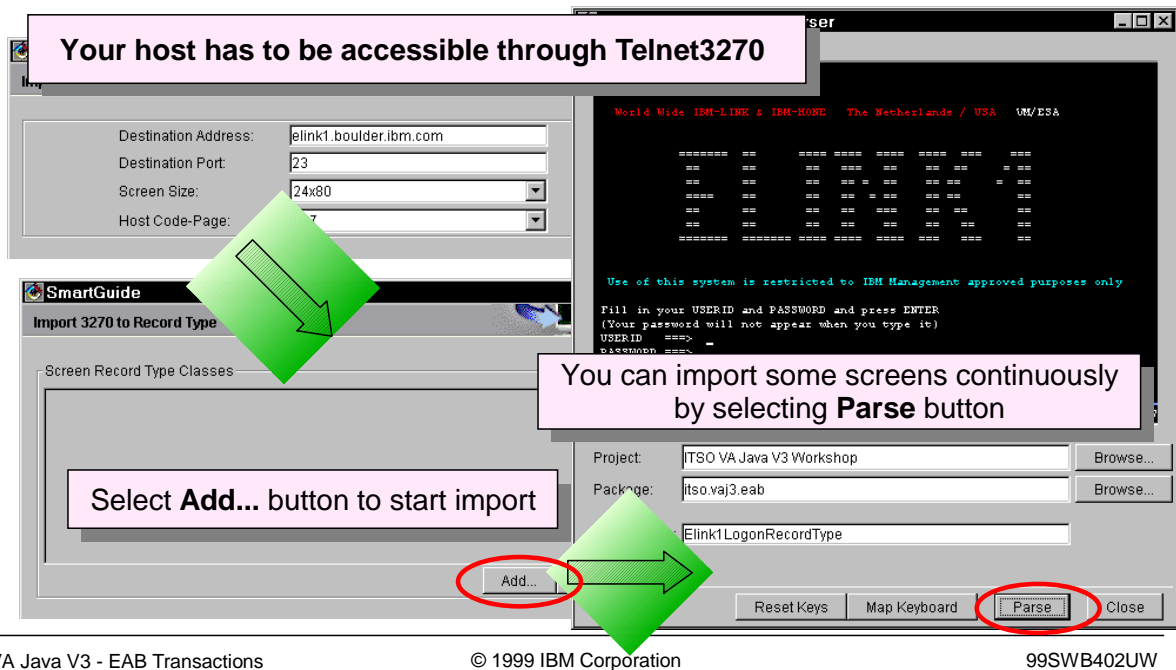


Figure 14-13. Import 3270 to Record Type (1)

The new imported from 3270 screen uses Telnet to access the host system. You then parse the screen that is currently shown.

## Import 3270 to Record Type (2)



In many cases, you have to edit imported record types

- **Editable field**

- Data in the fields will be stored as its *initial* values
  - When you create an instance of a record, fields are initialized using initial values
- If you want to use another value as its initial value, you *may* change it using the Record Type Editor

- **Read-only field**

- Data in the fields will be stored as its *constant* values
  - Constant values are used to check if a record can accept data received from host
- If data in Read-only fields will change, you *must* clear its constant value using the Record Type Editor

- **Suppose you imported 2 screens** (one is for invoking a query transaction and the other is for viewing results)

- You should clear initial values of fields in which you entered query conditions
- You must clear constant values of fields in which results are shown
- If these screens have fields in which current date/time are show, you also have to clear constant values of these fields

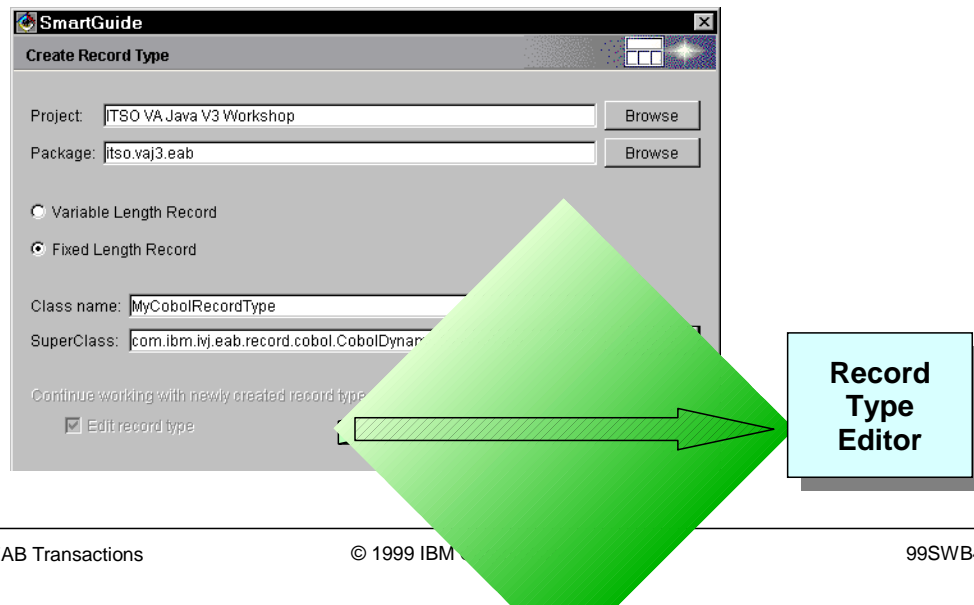
Figure 14-14. Import 3270 to Record Type (2)

## Create Record Type SmartGuide



**Using this SmartGuide, you can create a record type and start editing it**

- In VA Java V2, you have to create a record type using Create Class SmartGuide and launch Record Type Editor manually to edit it



*Figure 14-15. Create Record Type SmartGuide*

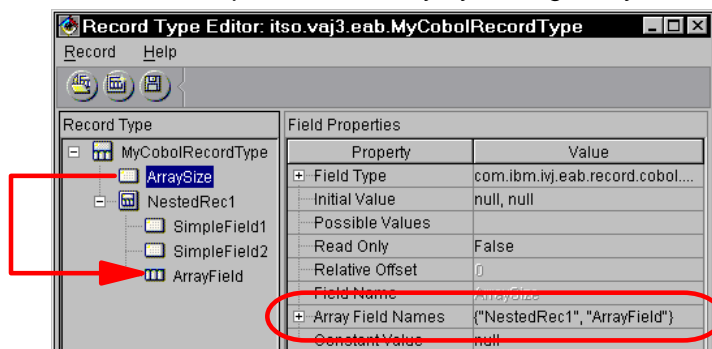
Version 3 makes it easier to create a new record type.

## Enhancements of Java Record (1)



### Java Record Framework supports variable size arrays

- To support "OCCURS DEPENDING ON", COBOL record implements this new feature
  - C Language Record Type also supports variable size arrays
- When you define a variable size array using the Record Type Editor, you have to :
  - Create and place an Array Size Control above an variable size array
  - Create and place an Array below the Array Size Control
  - Define the path to the Array by editing Array Field Names of the Array Size Control



**Custom Record style  
does not support  
variable size arrays**

Figure 14-16. Enhancements of Java Record(1)

## Enhancements of Java Record (2)



You can specify Possible Values to each field using values and/or ranges

Key	Value
Japan	"JP"
US	"US"
Europe	"EN", "FR", "DK"

Key Name      Possible Values

- By defining Possible Values, following methods are generated in your record

```

    • boolean is<KeyName>( <FieldAccessType> value )
    • <FieldAccessType> get<KeyName>()
    
```

- You can use these methods to verify field's values and get a possible value

```

if ( rec.isJapan( data ) ) {
    ....
} else if ( rec.isEurope( data ) ) {
    ....
} else {
    data = rec.getUS(); // Change to "US"
    ....
}
    
```

Figure 14-17. Enhancements of Java Record(2)

## Enhancements of Java Record (3)



### Object serialization support

- All records implement `java.io.Serializable` interface, and they are serializable
- Object serialization support is required to support EAB Session Bean Tool
  - When you call a remote method of EAB Session Bean, you pass a record as parameter
  - A remote method of EAB Session Bean return a record
  - EAB Session Bean is a Enterprise JavaBean (EJB), so parameters and return type of remote method have to be serializable
  - See "EAB Session Bean" for more details

*Figure 14-18. Enhancements of Java Record(3)*

## Enhancements of COBOL Record



### COBOL Record supports :

- java.math.BigDecimal as field access type
- Preferred type
  - You can specify a field access type for each field
  - This feature has been available for Terminal Record Type since VA Java V2
  - C Language Record Type does *not* support Preferred type
- Possible Values
  - LEVEL 88 item
- Variable size array
  - OCCURS DEPENDING ON

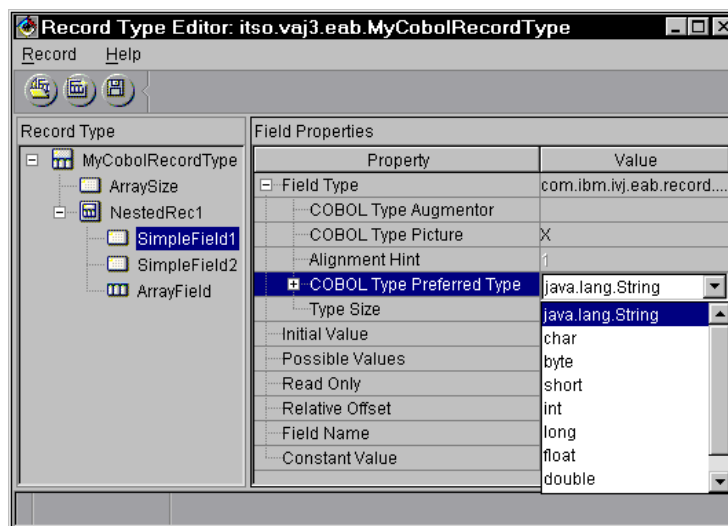


Figure 14-19. Enhancements of COBOL Record

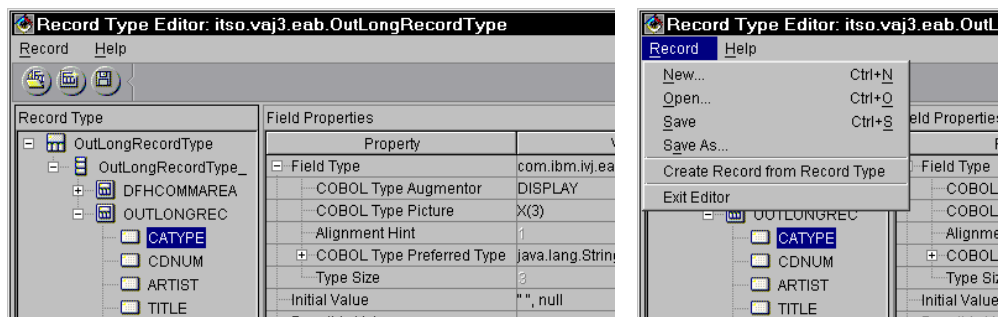
## New Record Type Editor



As you looked in previous pages, user interface of the Record Type Editor is enhanced

You can invoke following new operations from toolbar/menu

- Open an existing record type
- Create a new record type
- Save the editing record type as the same name or different name (save as)



- By using above features, you can easily create a new record type from existing record type

Figure 14-20. New Record Type Editor



## Create Record from Record Type SmartGuide



### The new SmartGuide provides you 2 new features

- Use Inner Classes
  - Depending on definition of a record type and code generation options, SmartGuide generates classes which represent subrecords
  - If you select **Use Inner Classes**, these subrecords are generated as inner classes of the generated record class
- Shorten Names
  - By selecting this, the SmartGuide makes the inner class names as short as possible

SmartGuide  
Create Record from Record Type

The records will be generated in:

Project Name: TSO VA Java V3 Workshop B

Package: tso.vaj3.eab B

Enter a class name for the records being generated:

Class Name: OutLongRecord

Select generation options:

Access Method: ☒ Direct ☐ Hierarchical

Record Style: ☒ Dynamic Records ☐ Custom Records

Additional Options: ☒ Generate with Notification

☐ Use Inner Classes

☐ Shorten Names

Property	Value
Floating Point Format	IBM
Endian	Big Endian
Remote Integer Endian	Big Endian
Code Page	037
Machine Type	MVS

Figure 14-21. Create Record from Record Type SmartGuide

## Create Command SmartGuide



VA Java V3 provides you a new SmartGuide to create EAB Command

**SmartGuide - Create Command**

Project:  Browse...

Package:  Browse...

Class name:

☒ Edit when finished → **Command Editor**

Connection spec:  
Class name:  Browse... Edit...

Interaction spec:  
Class name:  Browse... Edit...

► Specify ConnectionSpec and/or InteractionSpec  
► Modify their property values

**SmartGuide - Add Input/Output Beans**

Input record bean:  
☒ Implements IByteBuffer  
Class name:  Browse... Import...

Mapper class:  Browse...

Output record beans:  
☐ Use input bean as output  
☒ Select output record beans:

Output	Mapper	
OutLongRecord		

Add... Modify... Delete

► Specify a input record and/or output records  
► Specify mappers for the records  
► Modify property vales of the records

VA Java V3 - EAB Transactions

© 1999 IBM Corporation

99SWB402UW

Figure 14-22. Create Command SmartGuide

The creation of Commands has been greatly enhanced from Version 2.

Version 3 provides a SmartGuide that replaces the visual composition used in Version 2.

## New Command Editor



### Command Editor has brand-new user interface

- It become more efficient and easier to use
- It displays information better

- You can invoke following new operations from toolbar/menu
  - Open an existing command
  - Create a new command
  - Save the editing command as the same name or different name (save as)
  - Reset property values

You can migrate old commands by opening them with new editor

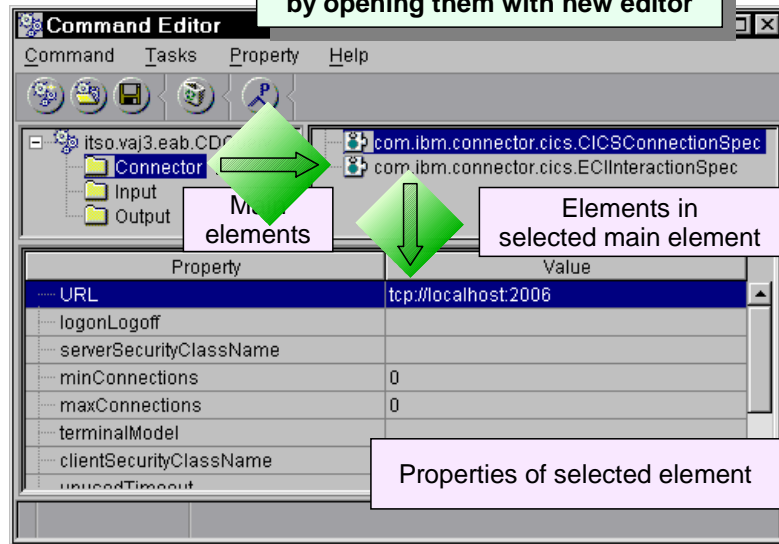


Figure 14-23. New Command Editor

## Limitation of New Command Editor



### VA Java V2

- You can edit an EAB Command using Command Editor and Visual Composition Editor (VCE)
  - By using special naming rules in VCE, you can keep compatibility with Command Editor

### VA Java V3

- *There is no compatibility between Command Editor and VCE*
  - If you created an EAB Command using Create Command SmartGuide or edited it using new Command Editor, you can *not* edit it using VCE
- You can still use VCE to edit EAB Command, but ....
  - You can create an EAB Command using Create Class SmartGuide and edit it using VCE
  - Using VCE, you can edit an EAB Command created in VA Java V2
  - If you migrate your EAB Command by opening it using new Command Editor, you *must* edit it using new Command Editor

**Use the new editor!**

Figure 14-24. Limitation of New Command Editor

In Version 3 you should not use visual composition for Commands. The new SmartGuide provides much better facilities.

## Changes in EAB Command/Navigator



### VA Java V2

- You can define input record as Bean or Bean Variable in Command Editor
- In Navigator, Commands which input record defined as variable will accept output record of previous Command/Navigator as input record

### VA Java V3

- You can *not* define input record as Bean Variable
- Instead, new Commands generated by Command Editor have **inputSetAtRuntime** property
- By setting **inputSetAtRuntime** property to `true`, new Commands work as the same as old ones which input record defined as variable
  - In Navigator, you have to set this property to `true`, if the new Command should accept output record of previous Command/Navigator as input record

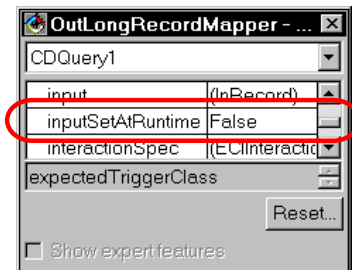


Figure 14-25. Changes in EAB Command/Navigator

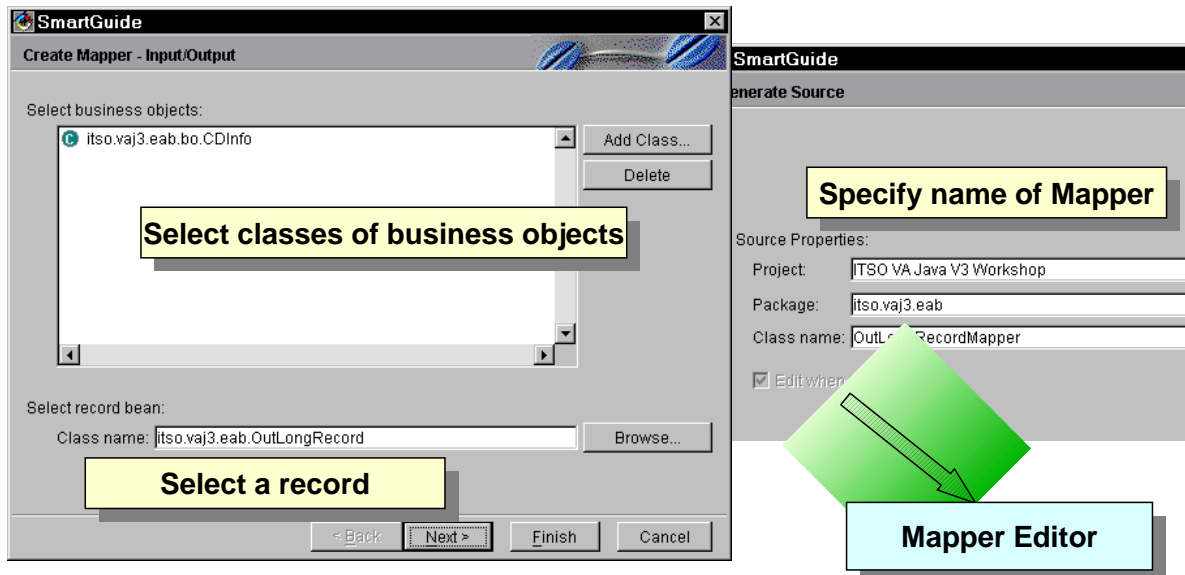
The Navigator has changed in Version 3 in the way the input bean is passed.

This must be considered when migrating applications to Version 3.

## Create Mapper SmartGuide



Using this SmartGuide, you can create a new Mapper and start editing it



VA Java V3 - EAB Transactions

© 1999 IBM Corporation

99SWB402UW

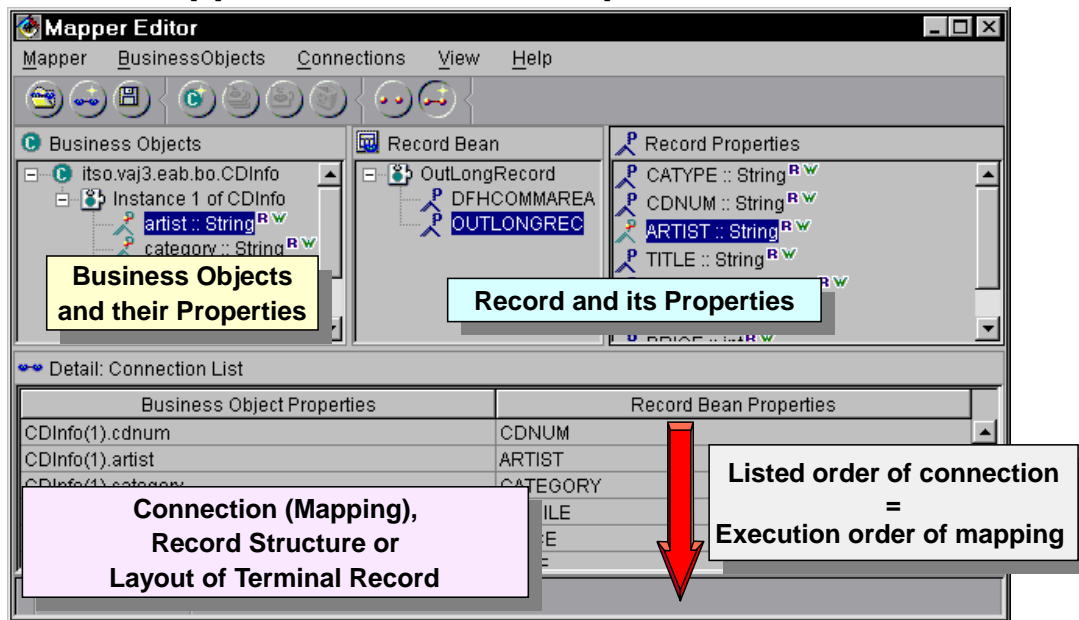
Figure 14-26. Create Mapper SmartGuide

The creation of a mapper is now also supported with a SmartGuide.

## Enhancements of Mapper Editor



Mapper Editor also has brand-new user interface and generates a Mappers that has new capabilities



VA Java V3 - EAB Transactions

© 1999 IBM Corporation

99SWB402UW

Figure 14-27. Enhancements of Mapper Editor

# Mapper: Array Support Enhancement



## Mapper and Mapper Editor support :

- Arrays of String and base types
- Nested arrays of business objects and records
  - e.g. A business object that has array of another business object
  - e.g. A record that contains an array of subrecords
- Array of business objects

**Mapping is applied to all elements of the array**  
e.g. You can *not* define different mapping for cust[0].name and cust[1].name

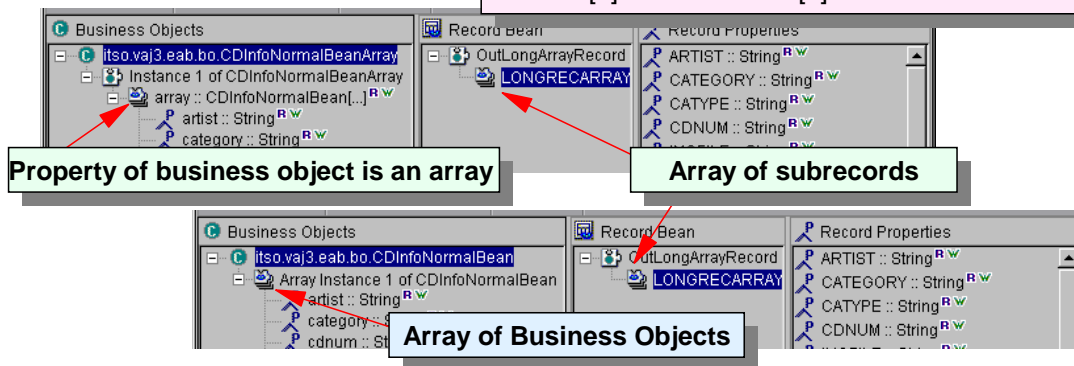


Figure 14-28. Mapper: Array Support Enhancement



## Mapper: More Enhancements



### Enhancement of Overlay and Nested Record Support

- Mapper and Mapper Editor support hierarchical access of properties and subrecords

### Code Generation Enhancement

- Mapper has accessor methods for business objects and record
  - You need not use `java.util.Hashtable` to set (non-key, non-EAB) business objects
  - You can promote there methods as properties of Command / Navigator
- Mapper provides new mapping methods
  - `java.util.Enumeration mapFromBusinessObjects()`
  - `java.util.Enumeration mapToBusinessObjects()`

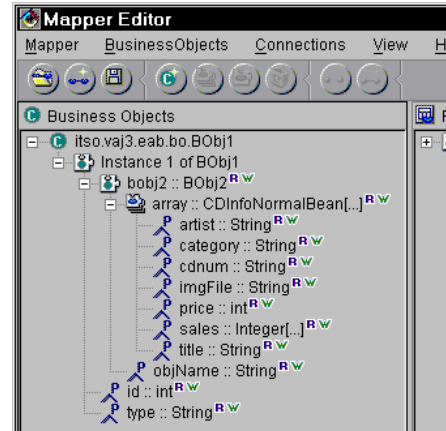


Figure 14-29. Mapper: More Enhancements

## EAB Session Bean Tool

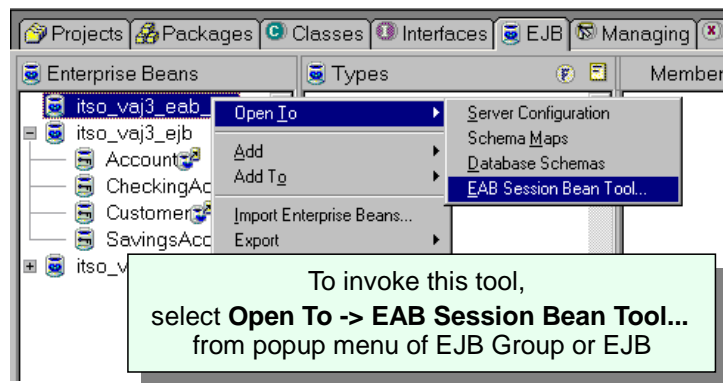


Using this tool, you can easily create an EJB, a so called EAB Session Bean

EAB Session Bean consists of a ConnectionSpec and some EAB Commands

Each method of EAB Session Bean :

- has a parameter that is passed to EAB Command as input record, if it has a input record
- executes a corresponding EAB Command
- returns a value that is an output record of EAB Command, if it has an output record



VA Java V3 - EAB Transactions

© 1999 IBM Corporation

Figure 14-30. EAB Session Bean Tool

With this new tool we can create a session EJB that invokes the EAB Command.

This allows, for example, so call the command from a JSP.

## Create EAB Session Bean SmartGuide (1)



Use this SmartGuide to create an EAB Session Bean

The figure shows three screenshots of the SmartGuide interface, illustrating the steps to create an EAB Session Bean.

**SmartGuide - Create EAB Session Bean**

Project:  Browse...

Package:  Browse...

Class name:

☒ Edit when finished

Connection spec:

Class name:  Browse...

Edit...

**► Specify a ConnectionSpec**

**► Modify their property values**

< Back Next > Finish Cancel

**SmartGuide - Add Methods**

Session bean methods:

Method
OutLongRecord queryAll(InRecord)
OutLongRecord query(InRecord)

Add... Modify... Delete

**► Add methods using Add... button**

Figure 14-31. Create EAB Session Bean SmartGuide (1)

## Create EAB Session Bean SmartGuide (2)



### Add methods to EAB Session Bean by :

- selecting already existing EAB Command
  - You can not use existing EAB Navigator
  - If a Command has no ConnectionSpec, ConnectionSpec of EAB Session Bean will be used
- specifying InteractionSpec, input record and output record(s)
  - Adding a method to EAB Session Bean is similar to creating EAB Command
  - ConnectionSpec of EAB Session Bean will be used

The SmartGuide dialog box is titled "SmartGuide". It contains the following fields and options:

- Method name:
- Method implementation:
  - ☐ Use existing command:
    - Class name:
    - Browse...
  - ☒ Specify method logic:
    - Interaction spec:
      - Class name:
      - Browse...
      - Edit...
    - Input record bean:
      - ☒ Implements IByteBuffer
      - Class name:
      - Browse...
      - Import...
    - Output record beans:
      - ☐ Use input bean as output
      - ☒ Select output record beans:
        - Table with columns: Output, Add..., Modify..., Delete...
        - Row 1: OutLongRecord

Figure 14-32. Create EAB Session Bean SmartGuide (2)

# EAB Session Bean Editor



## Using EAB Session Bean Editor, you can :

- add a new method to EAB Session Bean
- delete a method
- customize ConnectionSpec and InteractionSpec
- ....

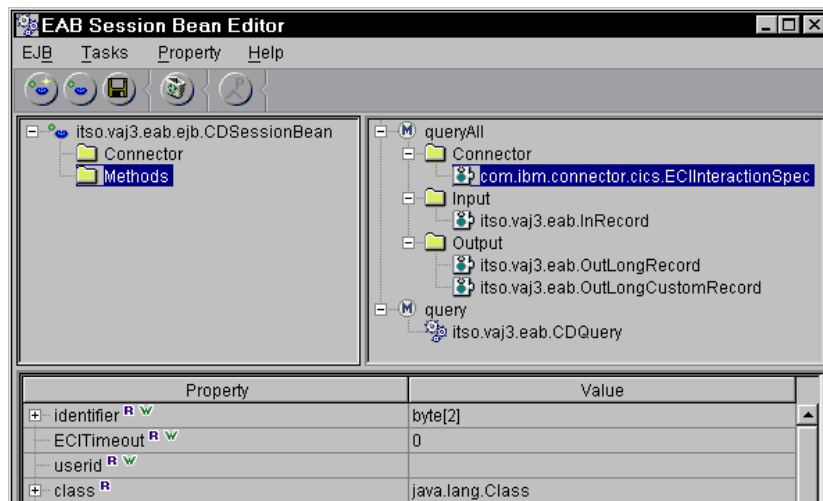


Figure 14-33. EAB Session Bean Editor

## Developing EAB Session Bean



**To develop EAB Session Bean, you have to do following development task by yourself**

- **Adding methods to remote interface**
  - You have to add methods which you added to EAB Session Bean using SmartGuide/Editor to remote interface
- **Setting up CCF runtime context**
  - CCF runtime context which EAB Session Bean use is set up and down in the following methods
    - private void beforeInteraction()      - Set up CCF runtime context
    - private void afterInteraction()      - Set down CCF runtime context
  - SmartGuide created these methods for you, but you have to change/add some code to these methods to set up and down CCF runtime context
- **Deploying**
  - This is the same with deploying another EJB
  - Make sure that CLASSPATH of your EJB container contains all jar files needed to execute EAB

*Figure 14-34. Developing EAB Session Bean*

## Environment Properties of EAB Session Bean



Customizations you have made to ConnectionSpec of EAB Session Bean are saved as environment properties

By changing environment properties at deployment time, you can easily customize ConnectionSpec

- For example, to change host name and port number, you need not edit source file of EAB Session Bean
- You may add/delete customizations to/from environment properties, because EAB Session Bean dynamically loaded them at runtime

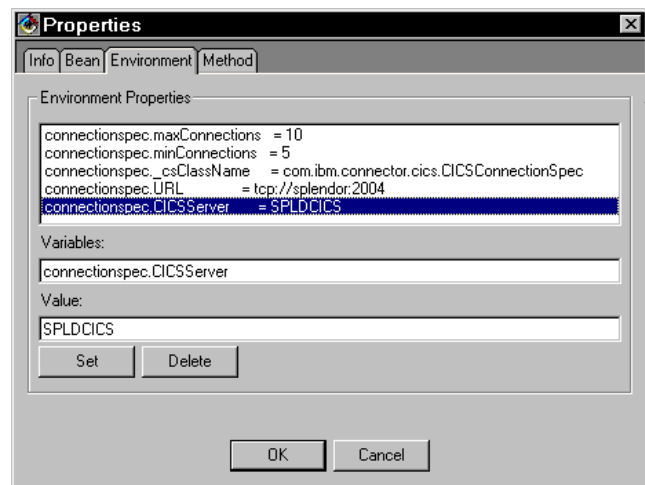


Figure 14-35. Environment Properties of EAB Session Bean

## EAB Entity Bean



**EAB Entity Bean is a BMP (Bean Managed Persistence) Entity Bean that implements persistency using EAB**

- For example, if your EIS has transactions which create, update, delete and query a customer information, you can create Customer EAB Entity Bean using these transactions

**You can create EAB Entity Bean using :**

- EJB Development Environment
- EAB (Enterprise Access Builder for Transactions)
  - Record
  - EAB Command and Navigator
  - Mapper

**But, VA Java V3 does not provides you special features to develop EAB Entity Bean**

*Figure 14-36. EAB Entity Bean*

It is also possible to create an EAB Entity Bean, with bean-managed persistence, that wraps around EAB commands.

However, VA Java provides no special support (SmartGuide) at this time.

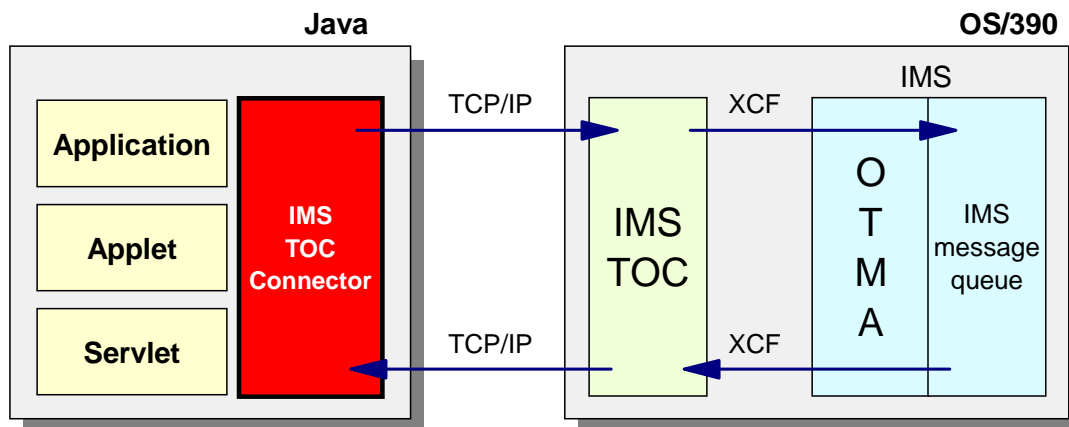


## IMS TOC Connector



**IMS TOC Connector provides you the way invoking IMS transactions through TCP/IP**

- IMS 5.1 or higher
- IMS TOC 2.1.3 or higher



**SG24-5427 IMS e-Business Connect Using the IMS Connector**

VA Java V3 - EAB Transactions

© 1999 IBM Corporation

99SWB402UW

*Figure 14-37. IMS TOC Connector*

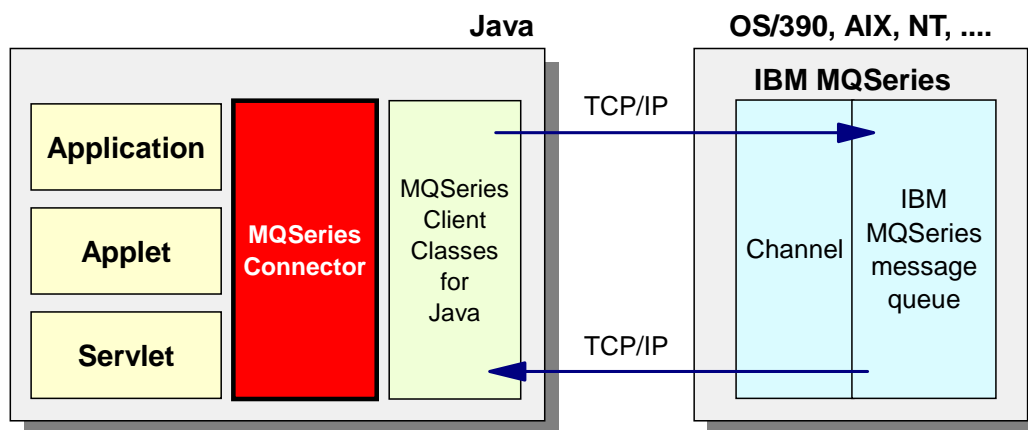
The IMS Connector was introduced in 4Q/98.

## MQSeries Connector



**Using MQSeries Connector, you can get/put a message from/to IBM MQSeries message queue**

- Current MQSeries Connector is based on the MQSeries Client Classes for Java from MQSeries V5.1



VA Java V3 - EAB Transactions

© 1999 IBM Corporation

99SWB402UW

*Figure 14-38. MQSeries Connector*

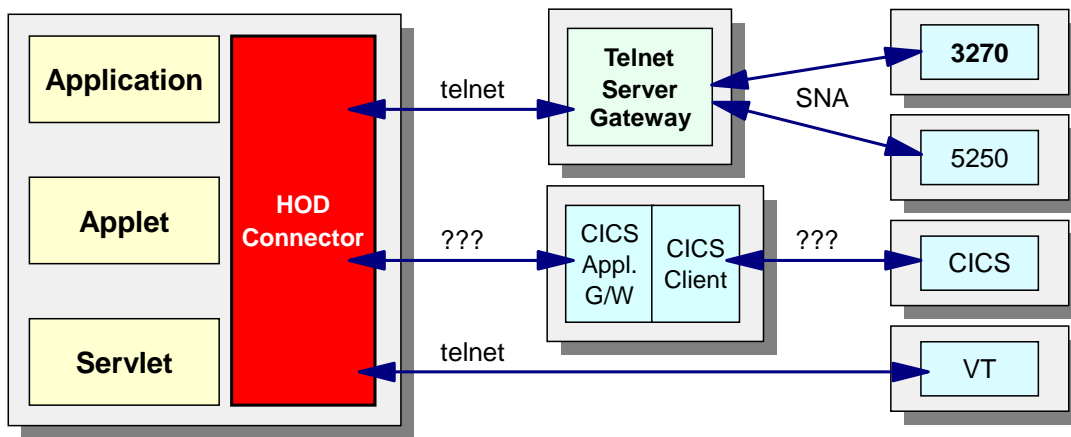
The MQSeries Connector was introduced in 4Q/98.

## Hot-on-Demand (HOD) Connector



**You can access to various systems based on a terminal**

- 3270, 5250
  - through the Telnet Server Gateway, such as IBM Communications Server family
- CICS
  - through the CICS Application Gateway and the CICS Client
- VT hosts



VA Java V3 - EAB Transactions

© 1999 IBM Corporation

99SWB402UW

*Figure 14-39. Host-on-Demand (HOD) Connector*

The HOD Connector was introduced in 4Q/98.

## SAP R/3 Connector - Access Builder



### Connector

### Access Builder

#### Common Connector Framework

- Connection pooling
- Common model (database, CICS, Encina, ...)
- Construct EJBs from commands with EAB tool

#### Integrated tool in VA Java

- BAPI Validated by SAP
- Separate BOR and RFC repositories
- Browse meta infos for BO and RFC
- Generate HTML documentation
- Embedded Javadoc documentation

#### Generate JavaBeans and EJBs

- JavaBeans for BOs, BAPIs and RFCs
- EJBs for BOs
- EAB commands for BAPIs and RFCs
- Exchangeable runtimes for server-side Java or 100% pure Java clients

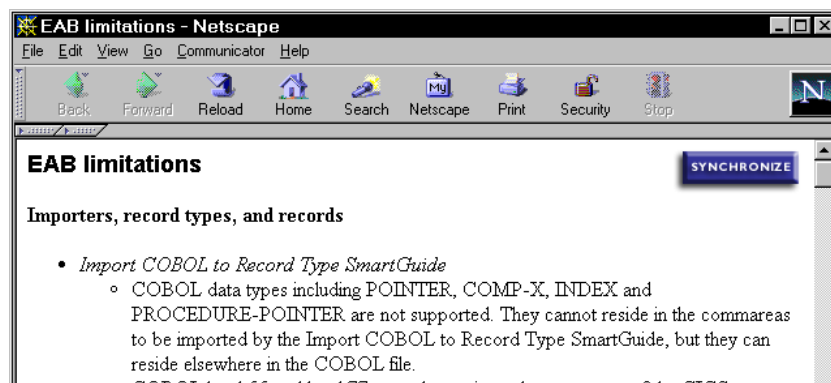
Figure 14-40. SAP R/3 Connector

# Limitations



**Before you use EAB tools, please check limitations described in VA Java help file**

- Components
  - > Enterprise Access Builder for Transactions
  - > Concepts
  - > Enterprise access
  - > Enterprise Access Builder for Transactions
  - > Limitations



VA Java V3 - EAB Transactions

© 1999 IBM Corporation

99SWB402UW

Figure 14-41. Limitations

## Summary



### New EAB provides you an easy way to access enterprise transactions using CCF connectors

- **SmartGuides**

- Create Record Type, Command and Mapper
- Import COBOL/BMS/FMS/3270 to Record Type
- Create Record from Record Type
- Create EAB Session Bean

- **Editors**

- Record Type, Command and Mapper
- EAB Session Bean

- **CCF Connectors**

- CICS connector [ECI/EPI]
- Encina DE-Light connector
- IMS TOC connector
- MQSeries connector
- Hot-on-Demand (HOD) connector
- SAP R/3 connector

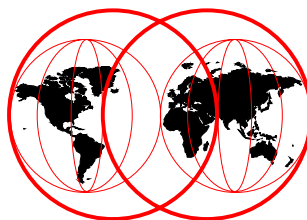
*Figure 14-42. Summary*

The Enterprise Access Builder for Transaction provides the link between Java applications and enterprise resources.

# **15 VA Java Version 3 Enhanced CICS Support with JCICS**

**VisualAge for Java Version 3**

JCICS Support



# Objectives



## Understand the JCICS Classes

### Requirements for JCICS

### ET/390 Setup

### Sample Program

- Coding
- Export and Bind
- Run the Program
- Debug the Program
- Visual Composition

### CICS IIOP

### CICS Connector

- Invoke one existing CICS transaction with input and output records (Commarea)

### JCICS

- Write CICS transactions in Java

### CICS IIOP

- Communicate from front-end Java with a CICS Java server program

Figure 15-2. Objectives

VisualAge for Java Version 3 provides a new way of accessing CICS applications: JCICS.

This support is in addition to the CICS Connector.

This material is based on a workshop by **Eugene Deborin** of ITSO San Jose.



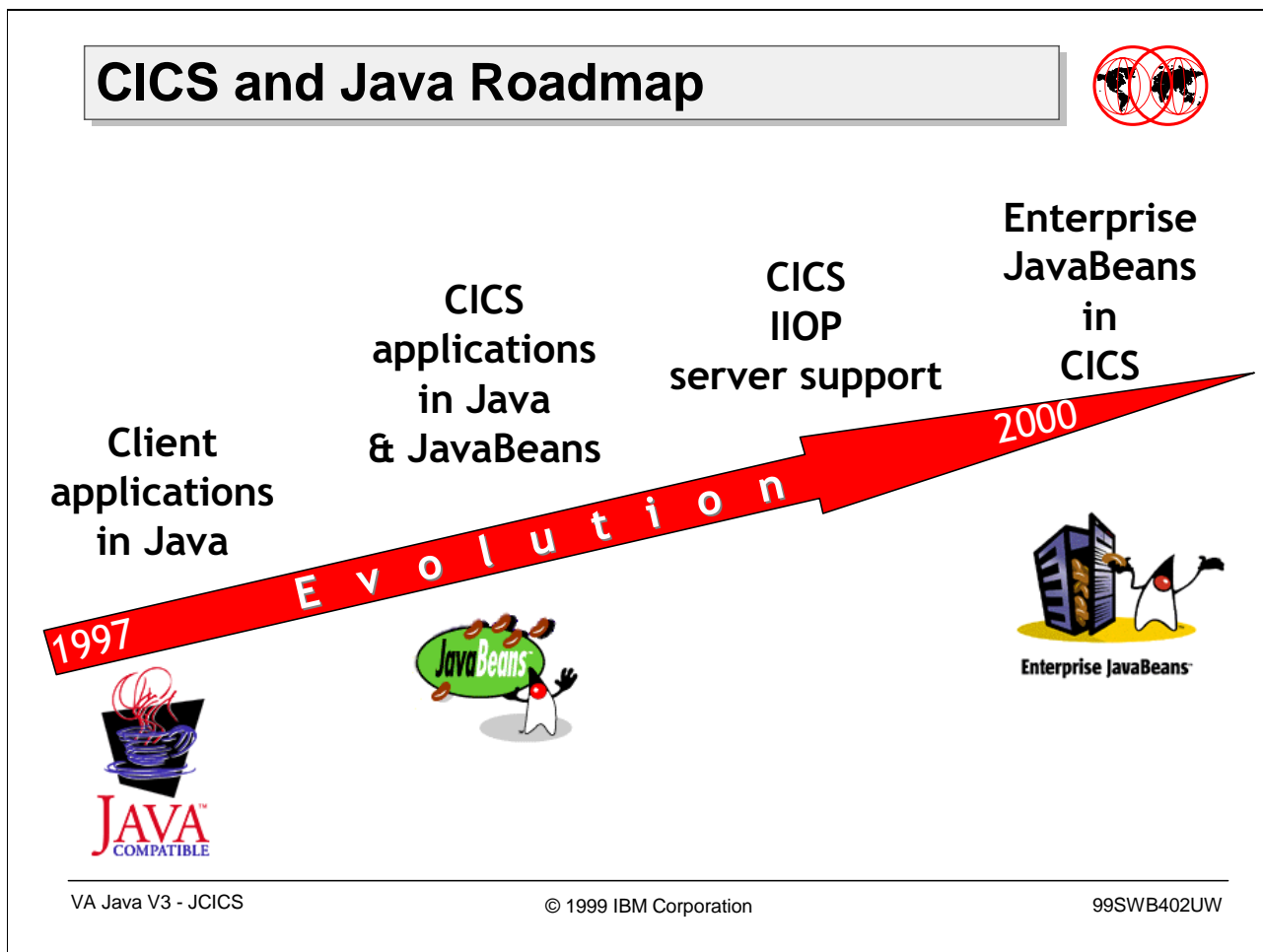


Figure 15-3. CICS and Java Roadmap

Today we can write CICS applications in Java with the CICS Connector, JCICS, and CICS IIOP support. Next year CICS will also support EJBs.

## Running CICS Java Programs



### Java programs in CICS can run in two ways ...

- Inside a real Java Virtual Machine
  - OS/390 JVM runs under separate OTE TCB
  - Switch is made to CICS quasi-reentrant TCB as required
- As a native OS/390 compiled program
  - Compile program using VisualAge for Java Enterprise Toolkit/390 Bytecode Binder aka HPJ
  - Result of compilation is stored in PDSE
  - Runs as an LE-enabled language (appears similar to C++)
  - Not all CICS Java programs are suitable for compilation

**HPJed applications currently show much better performance than those running under the JVM**

*Figure 15-4. Running CICS Java Programs*

CICS Java programs can be interpreted or compiled (using HPJ Compiler).

## JCICS Classes



### JCICS Classes provided as feature of VA Java

- contain CICS Java API classes used for development

### JCICS is the only way to call CICS services

- Are no EXEC CICS style interfaces for Java ..
- .. and hence there is no need to use a CICS translator with Java

### JCICS does not support all CICS services

- Terminal classes exist but there is no support for BMS
- No classes to support ..
- No CICS storage management classes exist - normal Java memory management should be used

- Transaction Types
- API classes
- Resource Classes
- Support for JavaBeans
- Error Handling
- Support for Java Record Framework

*Figure 15-5. JCICS Classes*

The JCICS classes provide a new API for Java.

## Requirements



- *CICS TS 1.3*
  - *OS/390 2.5+, SMP/E, RACF*
  - *MVS logging function (DASD logging function (single image), or a coupling facility)*
- *JDK 1.1 for mainframe Java compiles*
- *VisualAge for Java Enterprise for OS/390*
  - *Ensure OS/390 LE and OS/390 DFSMS are current*
- *ET/390 installed on workstation*
- *For remote debug of CICS Java programs*
  - *Customize CICS TCP/IP Socket Interface*
- *NFS Client for remote binding and deployment*

Figure 15-6. Requirements

# Simple JCICS Program



## Developing a simple program

- Create package and class
- Create a method:
  - *Get transaction id*
  - *Get APPLID*
  - *Get user id*
  - *Get terminal id*
  - *Write the data to the terminal*

*Figure 15-7. Simple JCICS Program*

## Simple JCICS Program Code



```
Source
public static void main(CommAreaHolder ca) {
    // Insert code to start the application here.
    Task myTask = Task.getTask();
    String sTranid = myTask.getTransactionName();
    String sRegionApplid = Region.getAPPLID();
    String sUSERID;
    try {
        sUSERID = myTask.getUserID();
    } catch(InvalidRequestException e) {
        myTask.out.println("The "+sTranid+
            " transaction should be run at a terminal.");
        return;
    }
    String sTerm;
    Object myFacility = myTask.getPrincipalFacility();
    if (myFacility instanceof TerminalPrincipalFacility) {
        TerminalPrincipalFacility term = (TerminalPrincipalFacility)myFacility;
        sTerm = term.getName();
    } else {
        myTask.out.println("The "+sTranid+
            " transaction should be run at a terminal.");
        return;
    }
    myTask.out.println(sTranid+"000I You are user "+sUSERID+" at terminal "+
        sTerm+" on region "+sRegionApplid+".");
}
```

Figure 15-8. Simple JCICS Program Code

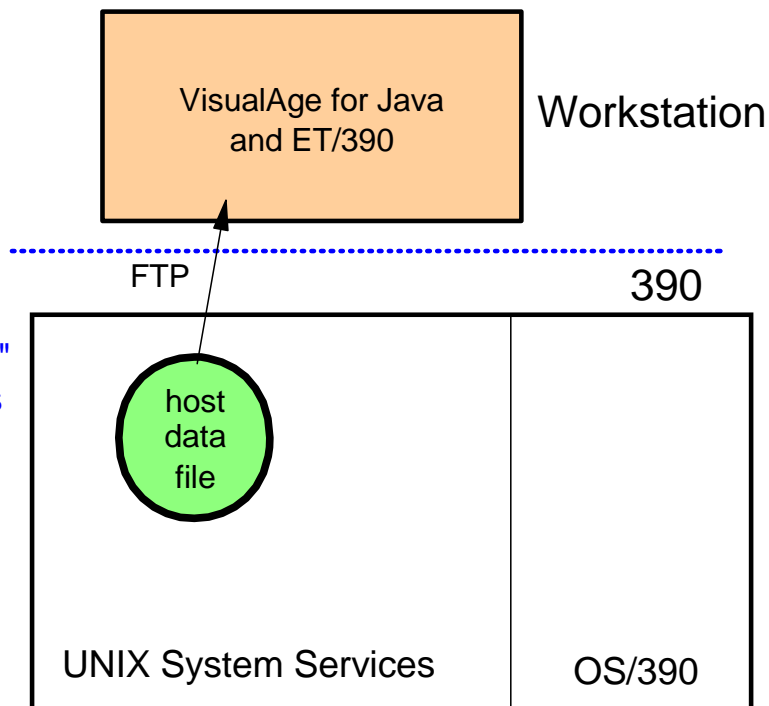
## ET/390 Setup



One time setup:

The host data file contains information needed by ET/390.

The host data file is FTP'd "under the covers" while you fill out a series of ET/390 configuration panels. ET/390 configuration is done for the workspace (one time), project, and package.



VA Java V3 - JCICS

© 1999 IBM Corporation

99SWB402UW

Figure 15-9. ET/390 Setup

To use JCICS on a /390 system you must setup the enterprise toolkit (ET/390).

## javaInstall File



```
@ @HPJHostName: wtsc61oe.itso.ibm.com
@ @HPJHome: /usr/lpp/hpj
@ @HPJBinderExecutablesPDSE: HPJ.SHPJMOD
@ @HPJBinderMessagesPDSE: HPJ.SHPJMOD
@ @HPJLERuntimeBind:
CEE.SCEELKED:CEE.SCEELKEX:CEE.SCEE OBJ:CEE.SCEE CPP
@ @HPJLERuntimeRun: CEE.SCEERUN
@ @HPJRuntime: HPJ.SHPJMOD
@ @HPJDebugger: EQA.V1R2M0.SEQAMOD
@ @HPJProfiler: CBC.SCTVMOD
@ @HPJJavaHome: /usr/lpp/java16/J1.1
@ @HPJPICLHome: /usr/lpp/hpj/jdebug/engine/jdebug.jar
@ @HPJCICSRegion: IYKC54:SCSCPAA6
@ @HPJCICSEXCi: CICSTS13.CICS.SDFHEXCI
```

/usr/lpp/hpj/javaInstall.data

Figure 15-10. JavaInstall File



## VA Java Host Sessions (1)



**ET/390 Add Host Session** **Tools -> ET/390 -> Host sessions**

Enter the following information to retrieve ET/390 Java Install Data from the host.

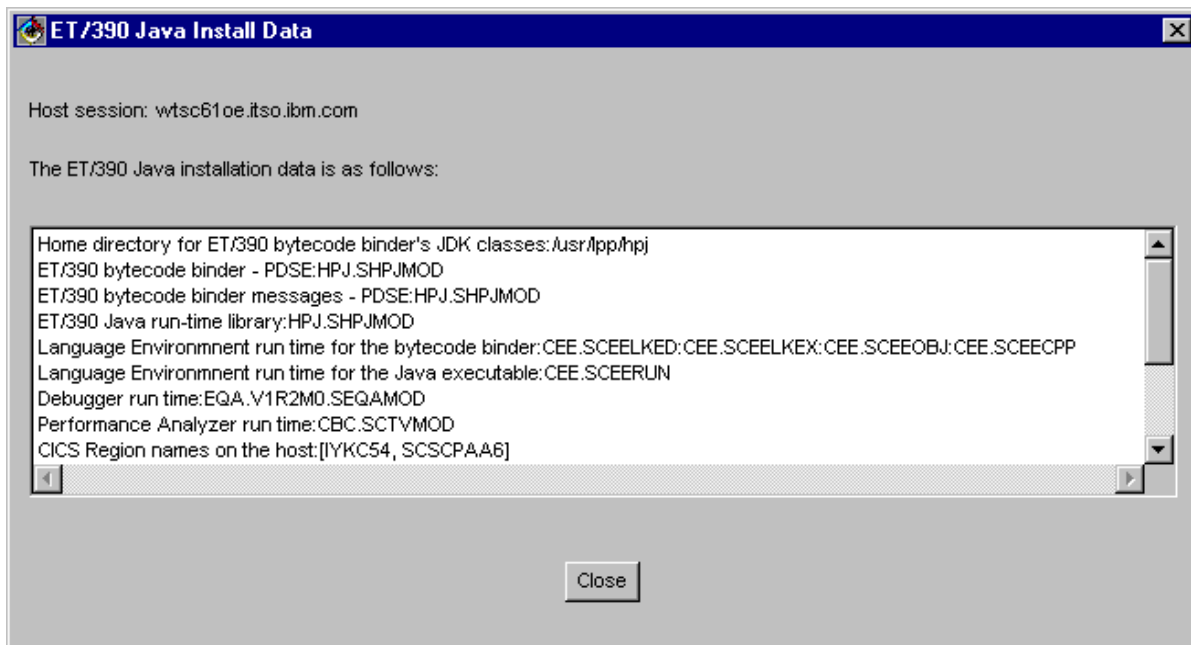
Host session:	<input type="text" value="tsc61oe.itso.ibm.com"/>
TCP/IP address:	<input type="text" value="tsc61oe.itso.ibm.com"/>
ET/390 Java Install Data file:	<input type="text" value="/usr/lpp/hpj/javaInstall.data"/>
FTP ID:	<input type="text" value="cicsrs5"/>
FTP password:	<input type="password" value="*****"/>

Retrieve the install data before selecting Add.

Press the Retrieve button to ftp a copy of /usr/lpp/hpj/javaInstall.data from the mainframe, then press the Add button

Figure 15-11. VA Java Host Sessions (1)

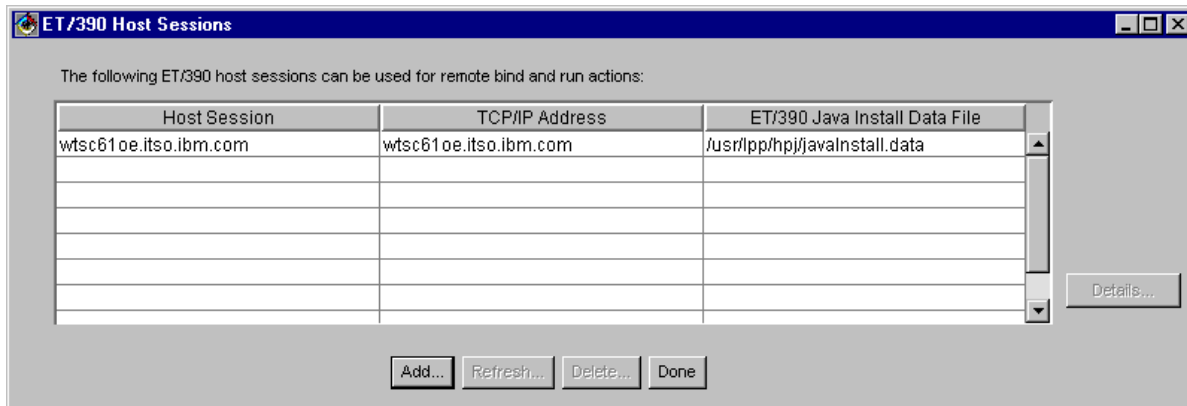
## VA Java Host Sessions (2)



### Results of /usr/lpp/hpj/javaInstall.data file

Figure 15-12. VA Java Host Sessions (2)

## VA Java Host Sessions (3)



You can have multiple host sessions

Figure 15-13. VA Java Host Sessions (3)

## ET/390 Logon Data



Tools -> ET/390 -> Logon data

Specify OS/390 Logon Data

Enter information for the remote logon to OS/390:

Host session ID: wtsc61oe.itso.ibm.com

User logon ID: cicsrs5

User logon password: \*\*\*\*\*

Save Cancel

Userid and password for rexec commands to  
OS/390 UNIX System Services  
to run HPJ compile and bind.

Figure 15-14. ET/390 Logon Data

# ET/390 Project Properties (1)



**ET/390 Properties for object CICSConf1**

Export and Bind Session

Information for ET/390 export and bind action:

Host session: wtsc610e.itso.ibm.com

Option set: ☒ Development ☐ Production

Bind CLASSPATH: /usr/app/cicsts/cicsta13/classes/dfjcics.jar

Directory for objects and lists: /u/cicrs5

Directory for executable or DLL: /CICSSYSF.OURJAVA.LOADLIB

Pre-bind script:  Browse...

Post-bind script:  Browse...

CICS region: SCSCPA6

Select the files to export and specify their destination:

☐ Class files ☒ Both Java and class files

Mount point on host: /u/cicrs5

Mounted directory for class files: R:\ Browse...

Local directory for Java files: S:\ Browse...

☒ Default to Parent / Use Local

OK Cancel

Tools -> ET/390 -> Project properties

VA Java V3 - JCICS

© 1999 IBM Corporation

99SWB402UW

Figure 15-15. ET/390 Project Properties (1)

## ET/390 Project Properties (2)



**ET/390 Properties for object CICSConf1**

Export and Bind Session  
    Bind Options  
    Advanced Bind Options  
Run Executable Session  
Run Main Session

**Bind Options**

Select the set of bind options:

☒ Development   ☐ Production

Code generation information:

☐ Rebuild all  
☒ Build a Java executable   ☐ Build a Java DLL  
☒ CICS application   ☐ Refresh CICS Program  
☐ Verify portability of code

Name for executable or DLL:

PDSE member name:

☐ Main class name

☒ Bind referenced classes  
☒ Generate compact code  
☒ Include Bind run-time options  
☐ Data for debug and trace   ☐ With hooks

Comment in program object:

Default to Parent / Use Local

OK Cancel

VA Java V3 - JCICS

© 1999 IBM Corporation

99SWB402UW

Figure 15-16. ET/390 Project Properties (2)

## ET/390 Project Properties (3)



ET/390 Properties for object CICSConf1

- Export and Bind Session
  - Bind Options
    - Advanced Bind Options**
    - Run Executable Session
    - Run Main Session

**Advanced Bind Options**

Select the set of advanced bind options:

☒ Development ☐ Production

Bind options to generate:

☐ Pseudo-assembler listing ☐ Offset from method

☐ Inline report

☐ Verbose information

☐ Object files only ☐ Objects in one directory

☐ Binder map to include: ALL

☐ Binder map cross-reference

Code for hardware architecture: 0 Tune: 2

☐ Optimize Maximum memory size:

Maximum spill area: 128

Other bind options:

Default to Parent / Use Local

OK Cancel

VA Java V3 - JCICS

© 1999 IBM Corporation

99SWB402UW

Figure 15-17. ET/390 Project Properties (3)

# ET/390 Package Properties



ET/390 Properties for object WHOPGM

Export and Bind Session  
Bind Options  
Advanced Bind Options  
Run Executable Session  
Run Main Session

### Bind Options

Select the set of bind options:

☒ Development ☐ Production

Code generation information:

☐ Rebuild all  
☒ Build a Java executable ☐ Build a Java DLL  
☒ CICS application ☐ Refresh CICS Program  
☒ Verify portability of code

Name for executable or DLL:   
PDSE member name:   
☒ Main class name:   
☐ Bind referenced classes  
☐ Generate compact code  
☐ Include Bind run-time options  
☐ Data for debug and trace ☐ With hooks

Comment in program object:

Default to Parent / Use Local

OK Cancel

VA Java V3 - JCICS

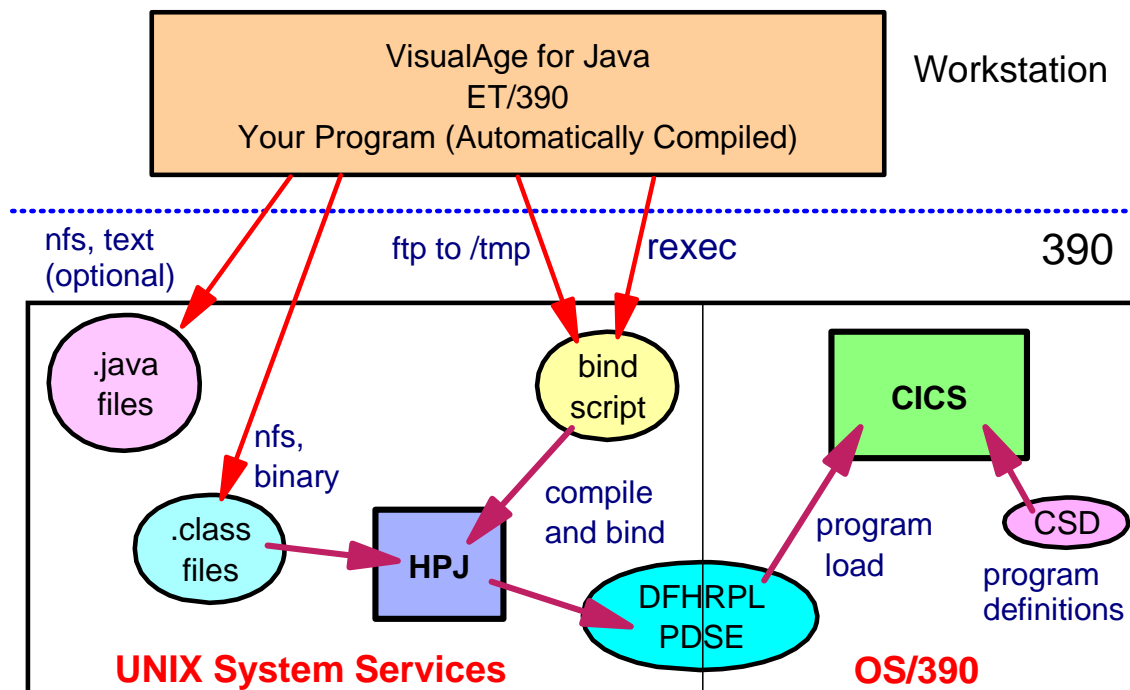
© 1999 IBM Corporation

99SWB402UW

Figure 15-18. Package Properties



## Export and Bind Flow



VA Java V3 - JCICS

© 1999 IBM Corporation

99SWB402UW

Figure 15-19. Export and Bind Flow

# Export and Bind



Tools -> ET/390 -> Export and Bind

```
***** Output from Remote Bind Command *****
IEW2278I B352 INVOCATION PARAMETERS -
TERM, AMODE=31, RENT, DYNAM=DLL, CASE=MIXED, COMPAT=CURR, MSGLEVEL=4
IEW2322I 1220 1 INCLUDE '/u/cicsrs5/WHOPGM/WhoPgm.mo'
IEW2322I 1220 2 ORDER 'RRM#exemaaindyn#C'
IEW2322I 1220 3 ORDER 'RRC#modinfo#C'
IEW2322I 1220 4 ORDER 'RRC#modulertestubs#C'
IEW2322I 1220 5 ORDER 'RRM#exemaaindyn#T'
IEW2322I 1220 6 ORDER 'RRC#modinfo#T'
IEW2322I 1220 7 ORDER 'RRC#modulertestubs#T'
IEW2322I 1220 8 INCLUDE '/u/cicsrs5/WHOPGM/WhoPgm.ro'
IEW2322I 1220 9 INCLUDE '/u/cicsrs5/WHOPGM/WhoPgm.o'
IEW2322I 1220 10 ORDER 'WHOPGM/WhoPgm#C'
IEW2322I 1220 11 ENTRY CEESTART
IEW2322I 1220 12 ALIAS 'WHOPGM/WhoPgm'
IEW2322I 1220 13 NAME 'WHOPGM'(R)

***** End of output *****
```

Output from bind returned to workstation

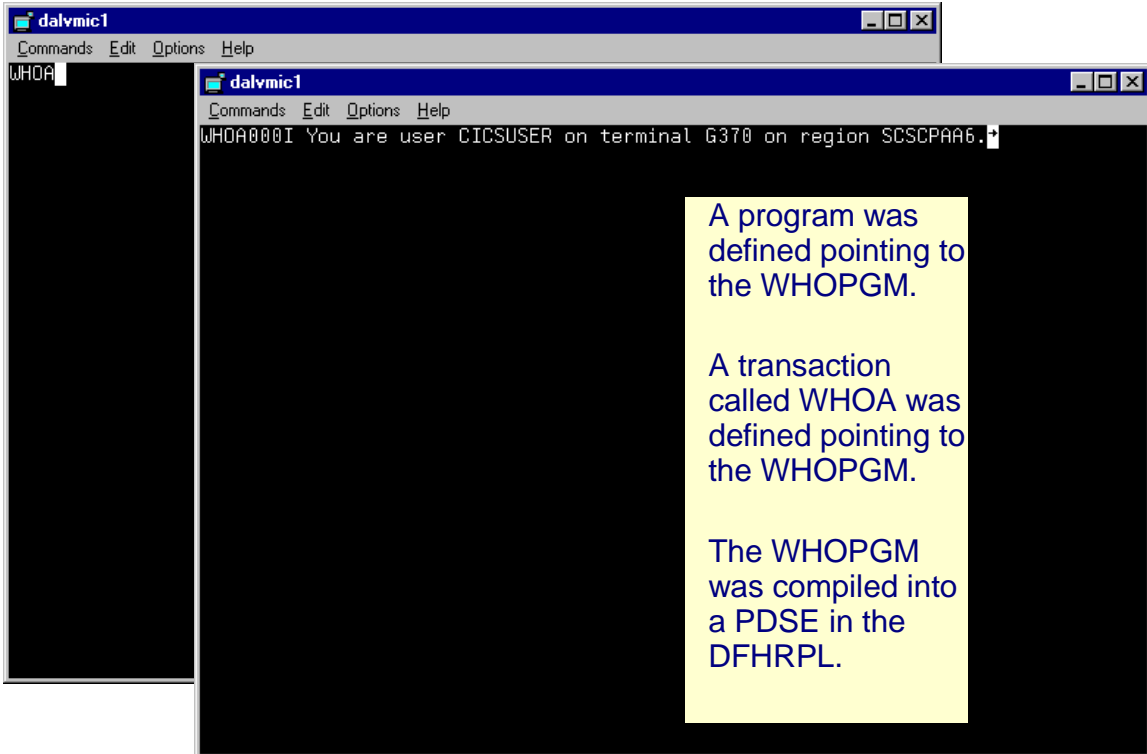
VA Java V3 - JCICS

© 1999 IBM Corporation

99SWB402UW

Figure 15-20. Export and Bind

## Executing the Program



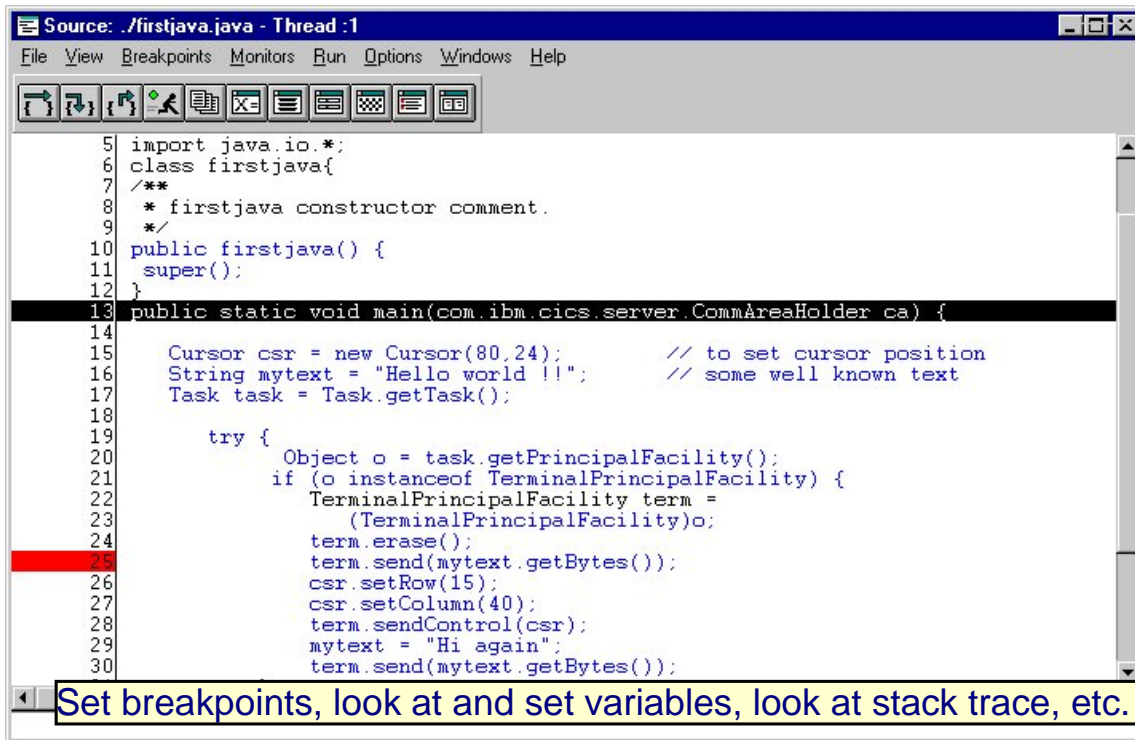
VA Java V3 - JCICS

© 1999 IBM Corporation

99SWB402UW

Figure 15-21. Executing the Program

# Debugging CICS Java Applications



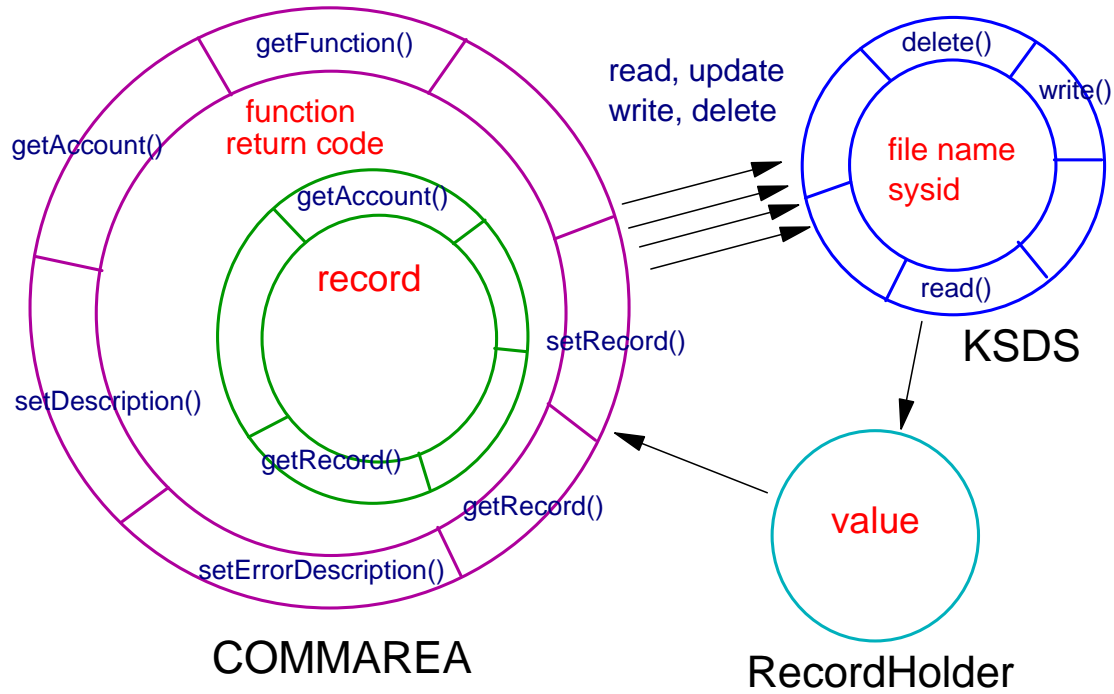
VA Java V3 - JCICS

© 1999 IBM Corporation

99SWB402UW

Figure 15-22. Debugging CICS Java Applications

## Visual Composition (1)



VA Java V3 - JCICS

© 1999 IBM Corporation

99SWB402UW

Figure 15-23. Visual Composition (1)

## Visual Composition (2)

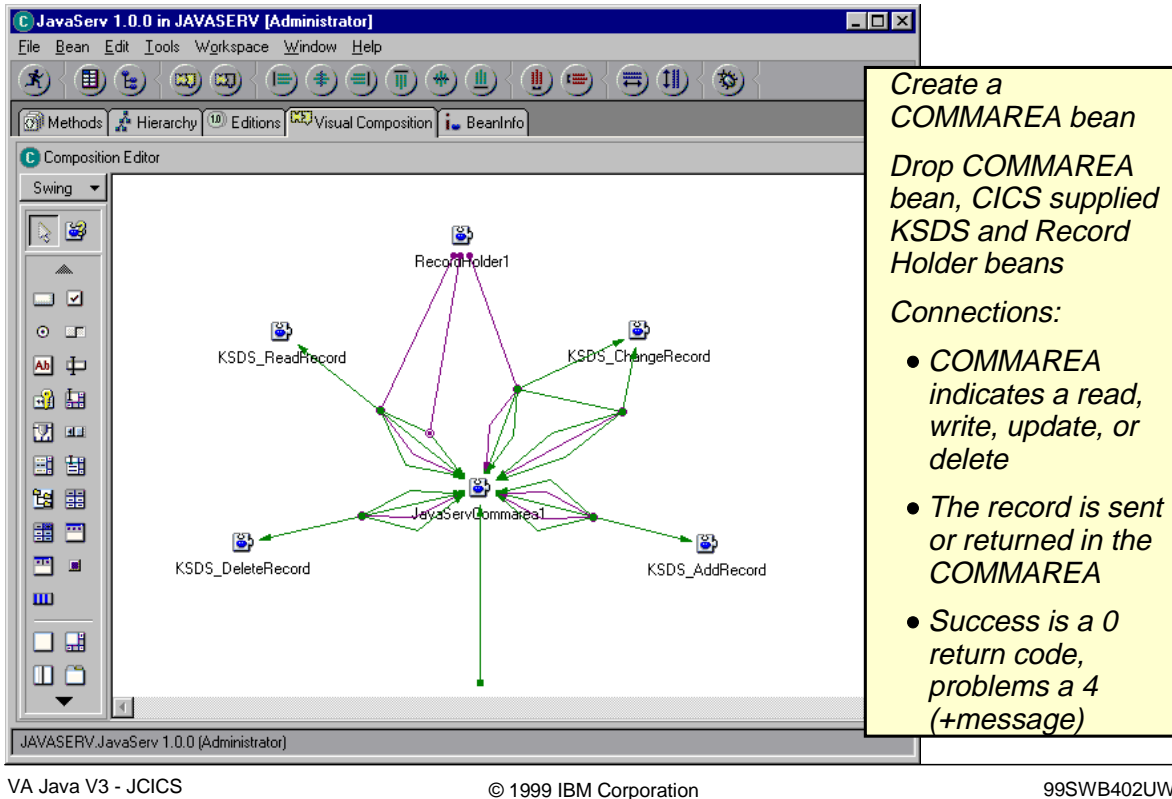


Figure 15-24. Visual Composition (2)

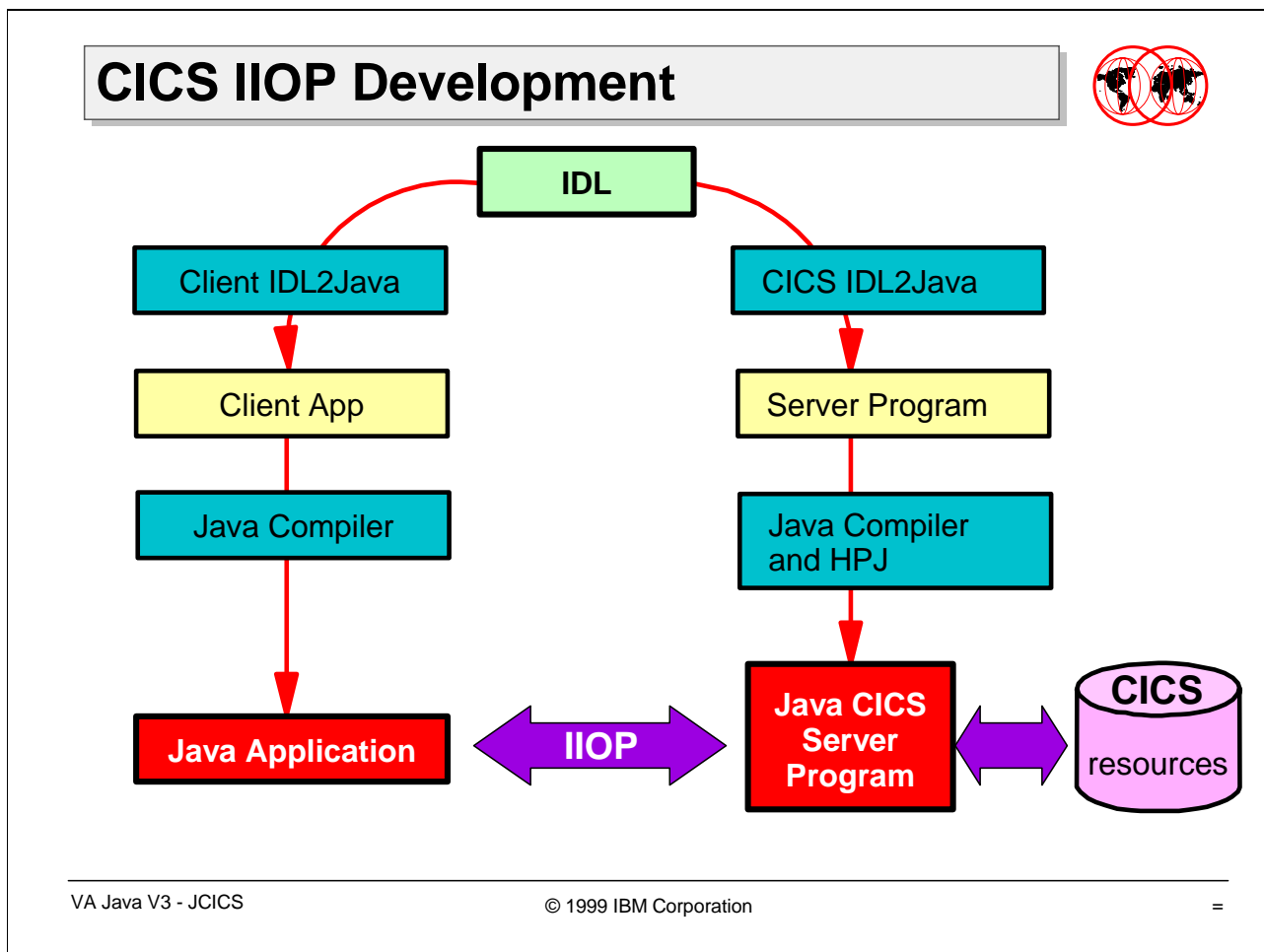


Figure 15-25. CICS IIOP

## More Information



### ● **SG24-5275 Java Application Development for CICS: Base Services and CORBA Client Support**

- CICS TS 1.3 manuals
- [www.software.ibm.com/ts/cics/platforms/internet/tgw30/ctgann30.html](http://www.software.ibm.com/ts/cics/platforms/internet/tgw30/ctgann30.html)
- [www.software.ibm.com/ts/cics/platforms/internet/cicsgw4j/index.html](http://www.software.ibm.com/ts/cics/platforms/internet/cicsgw4j/index.html)
- [www.software.ibm.com/ts/cics/platforms/client/clients301.html](http://www.software.ibm.com/ts/cics/platforms/client/clients301.html)
- [www.software.ibm.com/ts/cics](http://www.software.ibm.com/ts/cics)
- [www.software.ibm.com/ad/vajava](http://www.software.ibm.com/ad/vajava)
- [www.ibm.com/websphere](http://www.ibm.com/websphere)
- [www.ibm.com/java](http://www.ibm.com/java)
- [www.s390.ibm.com/java](http://www.s390.ibm.com/java)
- [www.JavaShareware.com](http://www.JavaShareware.com)
- [www.gamelan.com](http://www.gamelan.com)
- [www.javasoft.com](http://www.javasoft.com)

VA Java V3 - JCICS

© 1999 IBM Corporation

99SWB402UW

*Figure 15-26. More Information*

Read the redbook.

It is based on ET/390 and JCICS and VisualAge for Java Version 2, but should easily apply to Version 3 as well.



## Summary



- *VisualAge for Java*
- *Requirements*
- *JCICS Classes in VAJava*
- *Write a Small Program in VAJava*
- *ET/390 Setup (Workspace, Project, Package)*
- *Export and Bind*
- *Execute the Program*
- *Debugging*
- *Visual Composition*

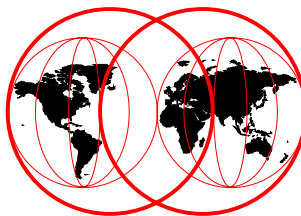
*Figure 15-27. Summary*



# 16 **VA Java Version 3 Extras (Technical Preview)**

**VisualAge for Java Version 3**

Extras - Tech.Preview



© 1999 IBM Corporation

99SWB402UW

## Technology Previews



### **JAX (Java Application Extractor)**

- Reduces application size

### **Jinsight**

- Java profiler/performance analyzer

### **XMI Toolkit and XMI Bridge**

- Bridge between Rational Rose and VA Java/Persistence Builder/EJB

### **Lotus XSL Processor**

- XML to HTML converter

### **WebDAV Client (WWW Distributed Authoring and Versioning)**

### **JavaPureCheck (from Sun)**

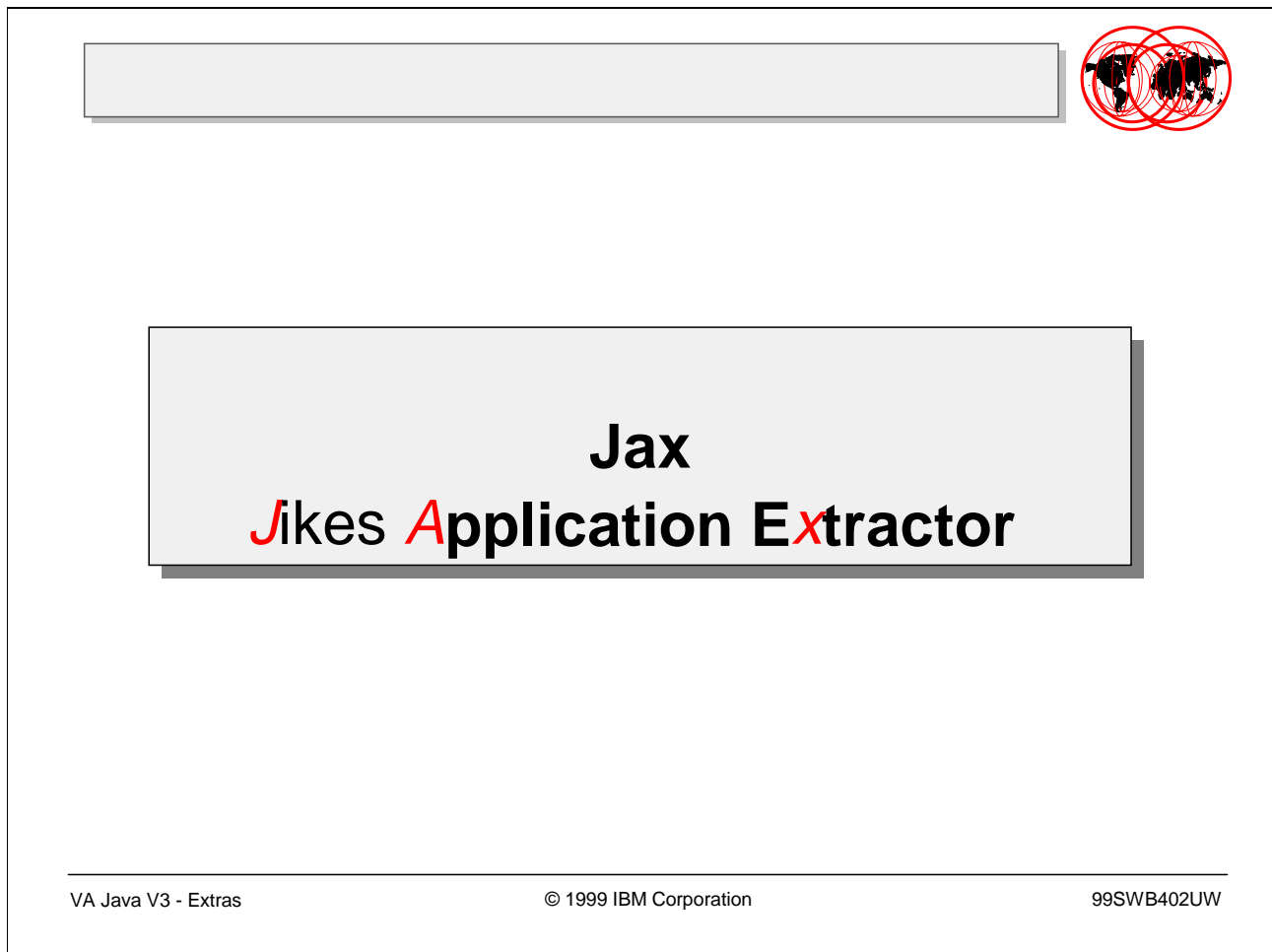
### **Rioja (Record I/O for Java Applications)**

- API extends the Record Framework of VA Java

### **XML Productivity Kit**

- Java Beans from XML data files, Wiring JavaBeans that process XML

*Figure 16-2. Technology Previews*



*Figure 16-3. Jax Jikes Application Extractor*

## Objectives (Jax)



### Understand the Concept of Jax

- Jax reduces the size of Java applications or applets
- remove unused (never referenced) code

### How to install Jax

### Use Jax from command line

### Use Jax from inside VA Java

Figure 16-4. Objectives (Jax)

## What is Jax?



**Jax stands for Jikes Application Extractor**

### What is Jikes

- Translates Java source files as defined in The Java Language Specification (Addison-Wesley, 1996) into the bytecoded instruction set and binary format defined in The Java Virtual Machine Specification (Addison-Wesley, 1996)
- Unlike other compilers, Jikes accepts the Java language only as specified: not as a subset, variant, or superset

### **Jax** reduces the size of your code

- extracts only the necessary classes, methods, and fields from existing java applications or applets
- applies optimizations to the code
- uses compression technologies
- the behavior of the code remains untouched

*Figure 16-5. What is Jax?*

## Advantages of Using Jax



### Low Requirements

- OS: any 100% Java 1.1 enabled platform
- Tools: JDK/JRE 1.1
- Needs < 800 K on your HD (incl. Documentation)
- Needs 40 K Ram in JVM for every class to optimize

### How good does Jax work?

- Typically size reductions are 30% - 90%  
(for large library based applications)
- IBM tested very large applications up to 2300 classes

*Figure 16-6. Advantages of Using Jax*



## JAX in Detail



### **Jax performs the following ordered steps:**

1. Removal of dead methods and fields
  2. Detection of live overridden methods
  3. Removal of unused classes and interfaces
  4. Inlining of methods
  5. Removal of non-essential attributes
  6. Shortening of internal method names and field names
  7. Removal of non-used entries in the constant pool
  8. Use compression technologies
- generate a \*.zip file for your code
  - generates a \*.log file for documenting changes

*Figure 16-7. JAX in Detail*

## Using Jax



**Jax is available on the VA Java 3.0 Enterprise CD Rom or on the alphaworks website ([www.alphaworks.ibm.com](http://www.alphaworks.ibm.com))**

### Using Jax from Command Line

- expand the extras\jax\jax53.zip file to a director
- java jax [command\_line\_options] <class\_name>

### Using Jax inside VA Java

- Import jax53.zip from extras\jax directory to a project "Jax"
- Check the class path for "jax::Default package for jax" and set "Extra directory path:" to the application/applet to be jaxed
- Click "Program" and set "Command line Arguments" to the name of the application/applet
- Run jax. Output files are generated under /IBMVJava/ide/project\_resources/ide/jax

*Figure 16-8. Using Jax*

## Summary (Jax)



**Jax reduces the size of your application**

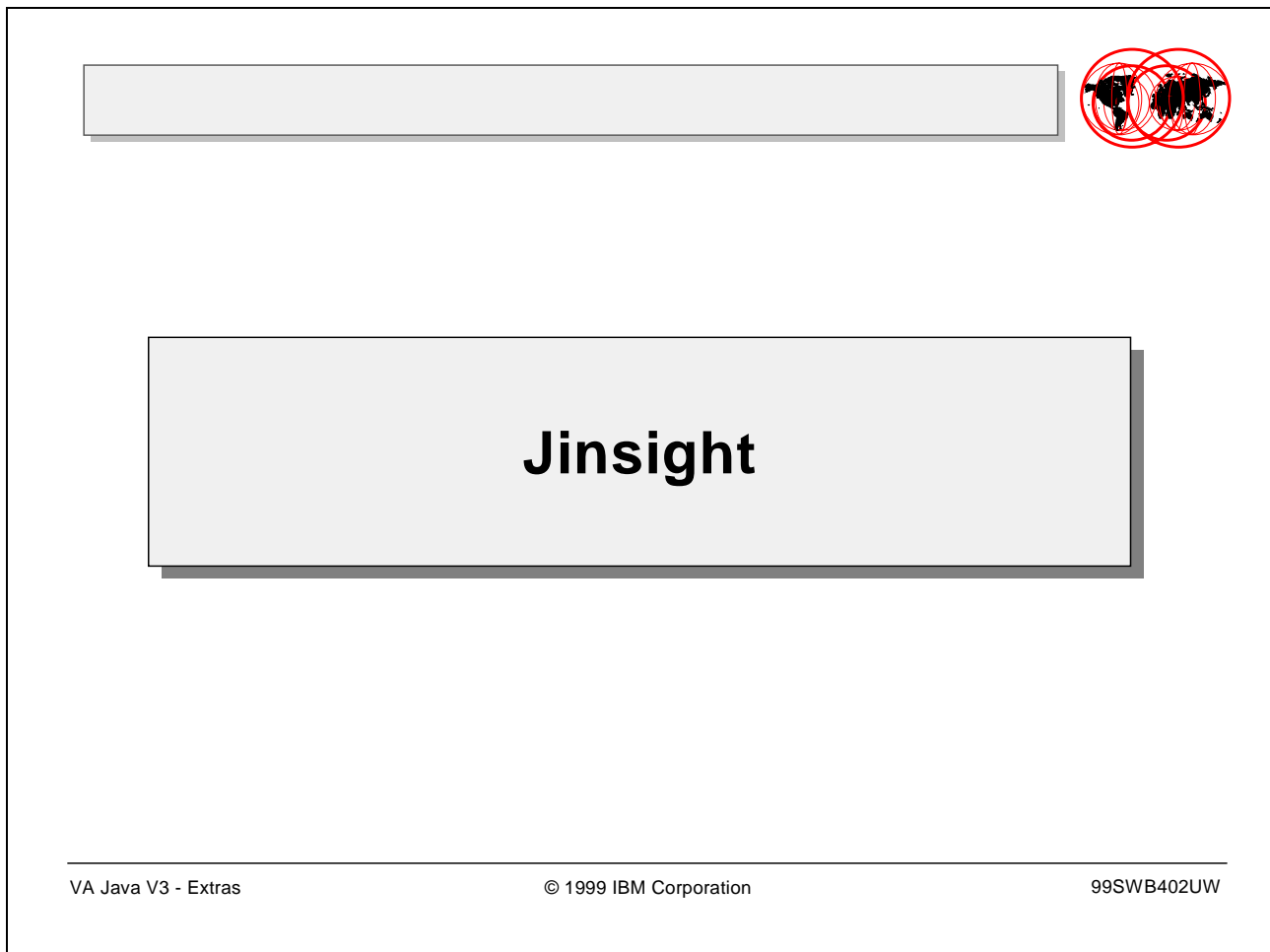
**Jax has low requirements**

**Jax can be used inside and outside VA Java**

### **More information:**

- JAX homepage(intranet) <http://w3.research.ibm.com/JavaTools/jax.htm>
- Jikes project (intranet) <http://w3.research.ibm.com/JavaTools/jikes.htm>
- The father of Jax (intranet) <http://w3.watson.ibm.com/~laffra/jax/>
- Cust.: please go to [www.alphaworks.ibm.com](http://www.alphaworks.ibm.com), enter Jax as search item
- Version 6.0 of JAX comes with a VA Java styled GUI

*Figure 16-9. Summary (Jax)*



*Figure 16-10. Jinsight*

## Objectives (Jinsight)



**Definition**

**Background overview**

**Installation**

**Using trace and visualization features**

**Uninstallation**

*Figure 16-11. Objectives (Jinsight)*

## Jinsight Definition



### Jinsight

- Debugging application
- Allows developers to view and analyze their code while a program is running
  - identify bottlenecks
  - track memory usage
  - weed out redundant code
  - object creation and garbage collection
  - execution sequence
  - thread interaction
  - object references
  - eliminate other performance - sapping flaws

**Designed specifically with object-oriented programs in mind**

*Figure 16-12. Jinsight Definition*

## Jinsight Background



### Current platform

- Tracing: Windows NT, Windows 95
- Visualizing: any Java platform,

**Is based on Jinight 1.1a from IBM alphaworks group and using additional instrumentation for Windows JDK 1.1.6 and 1.1.7b**

*Figure 16-13. Jinsight Background*

## Installation of Jinsight



### Exit VA Java V3 and run jinsight-tech-preview-install.exe

- When asked by the installation setup program provide the directory name where VisualAge for Java 3.0 is installed

### Restart VA Java V3

- Add a project named IBM Jinsight Tech Preview Trace Support
  - Import into it jar file api.jar and jinvatrc.jar from directory ide\program
  - This create the trace control program VAJavaTraceControl and an installation verification test program aTestPgm
- Add a project named IBM Jinsight Tech Preview visualizer
  - Import into it jar file jinsight.jar from directory Ide\program\Jinsight
  - This creates the visualizer program

*Figure 16-14. Installation of Jinsight*



## Creating a Program Trace



**Insight VisualAge, start the program VAJavaTraceControl**

**Insight VisualAge, start the program that you want to trace**

**Run your program to the point where you want to begin your observation, and press Begin tracing in the trace control panel**

- This will create a new trace file `ide\project_resources\jinsight.trc`

**Press End tracing at the end of the section you are tracing**

**Visualizing the results:**

- From VisualAge, start the IBM Jinsight Tech Preview visualizer
- Open file `jinsight.trc` and click Run.
  - You should see an indication that the visualizer is reading the trace
- When the end of the trace is reached, select Views to open one or more of Jinsight's views, and watch the results

*Figure 16-15. Creating a Program Trace*

## Uninstall Jinsight



### To return to using VisualAge for Java 3.0 without Jinsight Tech Preview

- Exit VisualAge for Java 3.0
- Copy file `ide\program\vm_orig.dll` to file `ide\program\ivjvm20.dll`
  - This restores the non-instrumented Java Virtual Machine
- Restart VisualAge for Java 3.0

*Figure 16-16. Uninstall Jinsight*

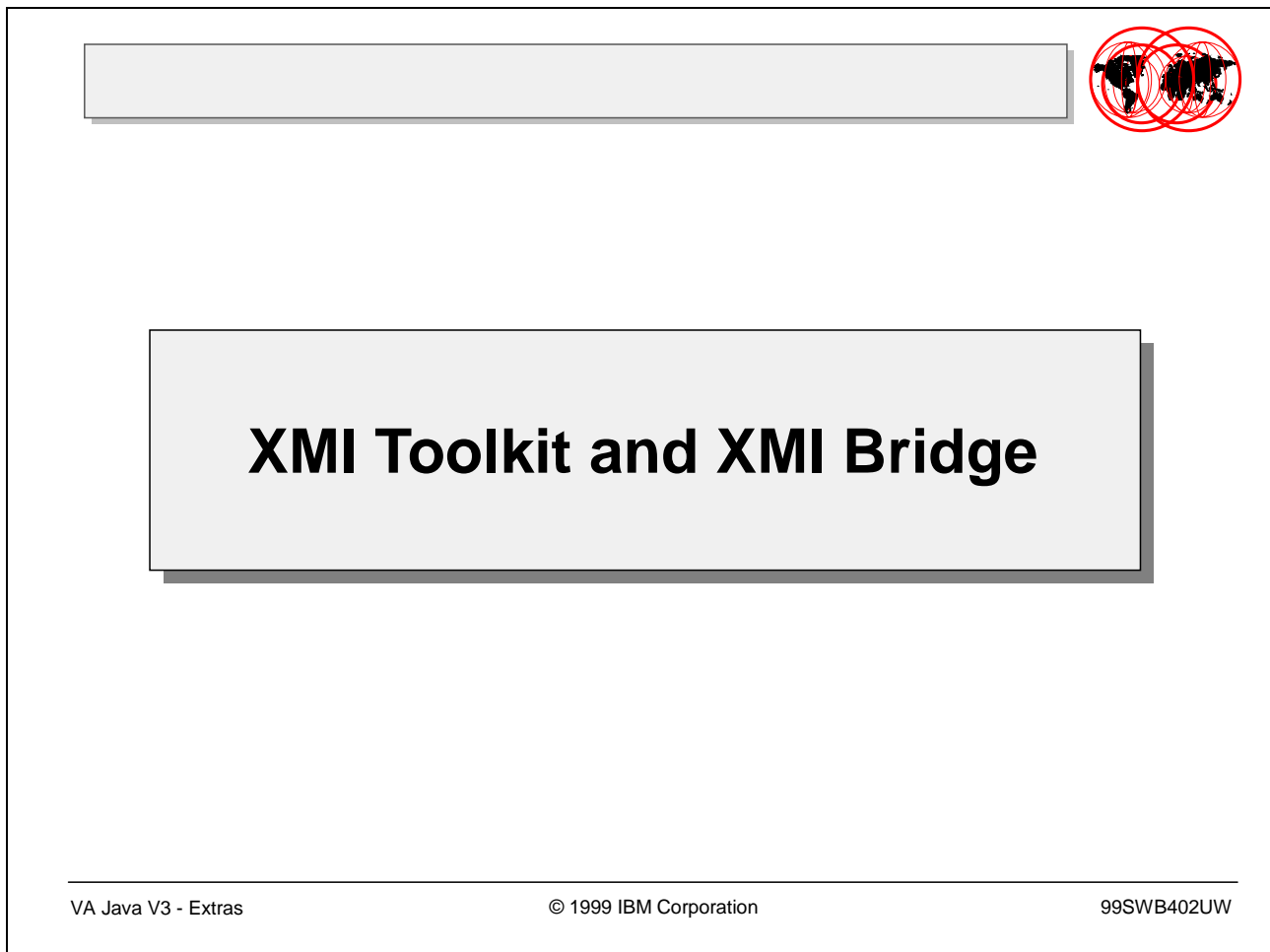
## Summary (Jinsight)



**We understand what Jinsight is and how to install and uninstall it**

**Using trace and visualization features**

*Figure 16-17. Summary (Jinsight)*



*Figure 16-18. XMI Toolkit and XMI Bridge*

## XMI Toolkit and XMI Bridge Overview



### XMI Toolkit

- Bridge between Rational Rose Model and VA Java Project

### **XMI Bridge:** Tech preview for Persistence Builder and Enterprise JavaBeans that enable you to import a Rational Rose model file or XMI document into a Persistence Builder model or EJB Group

- Import classes, attributes, associations, and inheritance structures from a Rational Rose model into a new or existing Persistence Builder Model
- Import classes, attributes, associations, and inheritance structures from a Rational Rose model into a new EJB Group
  - Each class becomes a Container-Managed Bean
  - Each attributes will have getter/setter methods in the Remote Interface
- Requires XMI Toolkit

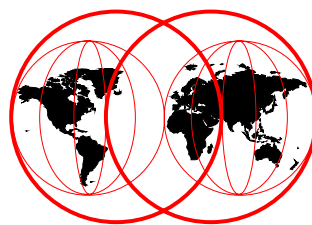
*Figure 16-19. XMI Toolkit and XMI Bridge Overview*



# 17 **VA Java Version 3 Java 2 Support**

**VisualAge for Java**

Java 2 Support



© 1999 IBM Corporation

99SWB402UW

# Objectives



## Overview

- Changes between JDK 1.1 to JDK 1.2
- Requirements for VA Java with JDK 1.2

## VA Java 3.0 Migration Steps

## VA Java V3 and VA Java J2

*Figure 17-2. Objectives*

The Java 2 preview version of VisualAge for Java (Professional) is available on

<http://www.software.ibm.com/vadd>



## Overview



### Major Changes between JDK 1.1 to JDK 1.2

- Swing classes (package names have changed)
  - Pluggable look and feel (easy assignment of OS look and feel)
  - Mouseless operation support
  - Drag and drop between Java-Java and Java-notJava
- CORBA Integration
  - Java IDL API
    - Add IDL-to-Java compiler generator
- Security Model
  - Policy based
  - Configurable and extensible access control model for users, groups and application
- RMI
  - Performance improvements
- JDBC
  - Support JDBC 2.0

*Figure 17-3. Overview*

## Overview (2)



### 3 platform implementation

- **Micro** - Complete and a highly optimized Java runtime environment for embedded systems
- **Standard** - provides a complete, secure foundation for building and deploying network-centric enterprise applications (Implemented in VA Java V3)
- **Enterprise** - extends mission critical enterprise applications to any Web browser

### Requirements for VA Java J2 with JDK 1.2

- Windows 95/98 or NT 4.0 with Service Pack 4.0
- Pentium Processor
- 64 MB RAM

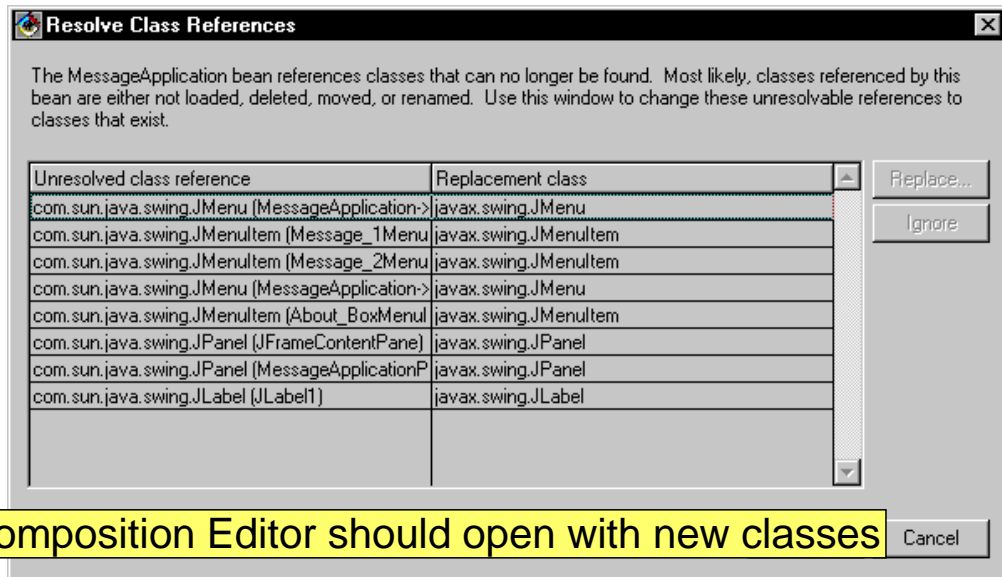
Figure 17-4. Overview (2)

## VA Java Migration Steps



### Import all the packages for migration Search and replace importing package

– com.sun.java.swing.\*; by javax.swing.\*;



Visual Composition Editor should open with new classes

Figure 17-5. VA Java Migration Steps

## Java 2 Support



### **VA Java ships Java 2 Support with J2 Edition**

- VisualAge for Java J2 Enterprise (also Professional)

### **Can install V3 and J2 on same machine**

- multiple install on same machine
- can run concurrently
- can share repository

### **Attention!**

- Swing classes are different
- DB2 JDBC drivers are different (see SQLLIB\java12)
- Tools have not been ported
  - Data Access Beans, Servlet Builder, Persistence Builder, EJB, etc.
- Can import V3 JAR files into J2

**Good approach: J2 GUI with V3 backend**

*Figure 17-6. Java 2 Support at GA*

## Summary



**You have an overview of the new features of Java 2**

**Know the migration steps for an application created using  
VA Java Version 2 (or 3) with JFC 1.03 to VA Java V3 with  
JDK 1.2**

**Can install VA Java V3 with JDK 1.1.7 and VA Java J2**

*Figure 17-7. Summary*



# Part 2 Exercises

## Exercise Preparation

These introductory instructions should make sure that VisualAge for Java is setup properly for the exercises.

1. Start VisualAge for Java.
2. Make DB2 JDBC drivers available: Select *Window -> Options*. Select the *Resources* page and add **c:\SQLLIB\java\db2java.zip**; to the *Workspace Class Path*. Click *OK*.
3. Load the following feature into the Workbench (*QuickStart*, select *Features -> Add Feature*), unless it is already in the workspace.
  - IBM WebSphere Test Environment
4. The other features we will load as required for each exercise.
5. Create a new project for all your code (*Selected -> Add -> Project*) and name it, for example, **ITSO VA Java V3 Workshop**. We will refer to this project as the **workshop project** in the exercises
6. Check that the sample code is available in the **c:\Va3Ws\sampcode** directory.
7. Prepare the WebSphere Test Environment:
  - Find the **SERunner** class (in **com.ibm.servlet**). Check the class path (*Run -> Check class path*) and select all the projects.
  - You will have to redo this when you add more projects to the Workbench and they are required for execution (for example, Servlet Builder).
8. Open a **DB2 Command Window** and connect to the **SAMPLE** database

```
db2 connect to sample
db2 select * from department
db2 select * from employee
db2 connect reset
```

If the database does not exist, create the database in DB2 by running the d:\SQLLIB\bin\DB2SAMPL.EXE file.

Now connect to the SAMPLE database as user **userid**:

```
db2 connect to sample user userid  
Enter current password for userid: password
```

If the connect fails, the user **USERID** must be created in Windows NT (with password = **password**) and authorized to the database using the **DB2 Control Center** (give all authorizations).

## Attention

**Save your workspace after every exercise!**



# E1 Using the IDE Enhancements

## What this Exercise is About

In this lab we want to use some of the new features of the IDE of VA Java 3.0

## What You Should Be Able To Do

At the end of this lab you should be able to

- ☐ Understand the IDE enhancements
- ☐ Create an application with the new Smart Guide for visual applications
- ☐ Using the global search & replace function
- ☐ Generate accessory methods for fields
- ☐ Create and using keyword completions and macros
- ☐ Maintaining project resources in VA Java
- ☐ Understand how the different Code Generation Options are working
- ☐ Use the filter options
- ☐ Use the new debugger functions.

## Introduction

We start with a simple application, which will be extended by using some of the new IDE features. Also we will use this application later to show some debugger enhancements.

---

## Exercise Instructions

### Part I: Using the new SmartGuide for Visual Applications

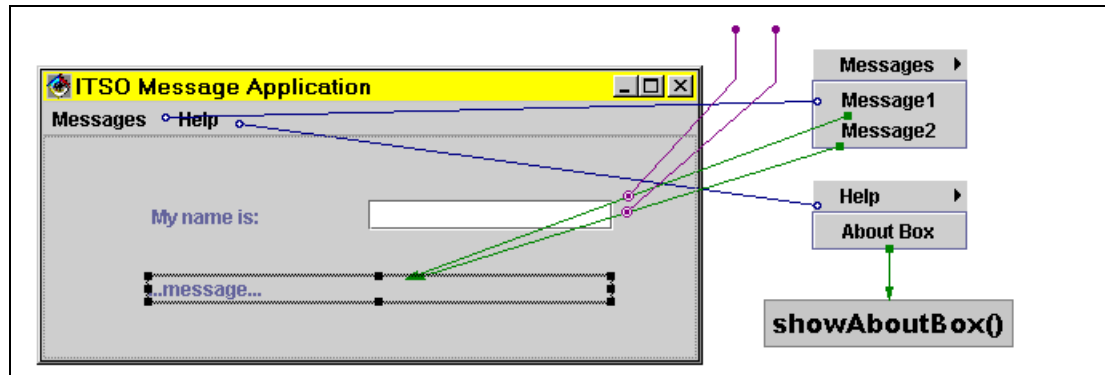
1. Create a new **itso.vaj3.ide** package in the workshop project.
2. Start the SmartGuide in the menu *File -> Quick Start -> Basic -> Create Application*.
3. Browse for the workshop project and itso.vaj3.ide package, and enter **MessageApplication** as the name of the application.
4. Select **Create Swing based application** and click on *Next*.
5. Change the Title Bar to **ITSO Message Application**.
6. Deselect **Toolbar**, **Statusbar**, and **Center and pack frame on screen**. Select **Splash Screen** and **About Dialog**.
7. Select **Menu Bar**, click *Details*
  - Remove **File**, **Edit**, **View** menus (Select, click *Delete*)
  - In the **Help Menu**, remove **Help Topics** and **Books Online**
  - **Add** a new menu called **Messages**
  - **Add** two Menu Items to this menu, called **Message1** and **Message2**
  - Move the Messages Menu to the top of the menu list (click *Up*)
  - Close the details dialog with **OK**.
8. Click *Finish*. Be patient, the generation of the application takes some time... Finally the Visual Composition Editor opens for the Message Application.
9. Run the application as is (click *Run* icon). See the splash screen (cow), use the *Help -> About Box* menu to see the MessageApplicationAboutBox. Check in the Workbench what classes have been generated.

### Complete the Application

10. Open the **MessageApplication**. On the **Members** pane, use the **Create Field SmartGuide** (F icon) to add a field called **message1**, type **String**, set initial Value "Message1", access modifier is private, enable the **Access with getter and setter methods**. Click *Finish*.
11. Manually create a field in the class declaration:

```
private String message2 = "Message2";
```

Select the **MessageApplication** class in the Workbench and select **Generate -> Accessors** from the context menu. Select message2 from the list, click *OK*. This creates getter/setter methods for the message2 field.
12. Insert two labels and one entry field. The text of the first label is **My name is:**, the text of the second label is **...message...**
13. Connect the menu item **message1**, **action performed** to the message label, **text**, and connect the parameter **value** to the property **message1** (on the free-form). Do the same for the **message2** menu option with the **message2** property as parameter value.
14. The layout is shown below:



15. Generate the code.
16. Go to the **Members** pane and study the code. How are the action event handled ?
17. Go to **Bean** -> **Code generation options**. There are now different ways to generate the event handling code. Change the option and generate the code again. See the difference when using inner classes versus no inner classes.  
  
You can change the default by using *Window -> Options*, then *Visual Composition -> Code generation*. Here you can also select **Generate meta data method** to have the visual composition saved in the source code as a method named **getBuilderData**.
18. Run the application and test the menu options.
19. Select the application panel (with the labels and entry field) and double-click to open the **Properties**. Change the **layout** from <none> to **GridBagLayout**. All positioning is kept, and new items can now be added. This is a nice improvement when designing GUIs. Generate the code. Close the visual composition.

## Part II. Search/Replace, Keyword Completion, and Macro Generation

20. Go to *Workspace -> Text Search/Replace*.
21. Type "Message1", **Replace** it with "Hello World from ...your name...".
22. Select *Working Set*, click *Choose*, then *New*. Select the **itso.vaj3.ide** package and the 3 classes, set the name for the working set to **itso.vaj3.ide**. Close the choose dialog.
23. Click *Search*, and then **Replace** when Message1 is found. Note that you find Message1 twice, for the field and for the menu item. Close the search window and the Search Results window.
24. Go to *Window -> Options -> Coding -> Keyword completions*.
25. Select the **if0{}** statement from the list and edit the **Completion** to:
 

```
if (<|>true){
    // this is a VA Java V3 IDE sample
}
```
26. Go to **Macros**, click **Add** for a new macro called **hello**. Set the **Expansion** to:
 

```
message2="Hello World two";
```

 Click *Apply*, and *OK*.
27. Go to the method `getMessage2`. Type inside the method body: `if (` and press **ctrl+Space**. In the popup-menu select the keyword **If0{}** and you see the completed keyword with your extension.

28. Type **hello** inside the statement block and press **ctrl+Space**. Your macro will be executed and hello extends to message2 = "Hello World from the ITSO";
29. Run the application and see what has changed.

### Part III. Use the Resource Management

30. Select the project ITSO VA Java V3 Workshop and select **Open To -> Resources**. You will find a number of GIF files that are used in the generated application. In the context menu you can see the actions Open, Delete, Rename.
31. The **Open** action uses a file association. Use *Window -> Options -> Resource Associations*. Enter the extension **.GIF**, click *Add*, then select **External Program** and click *Browse*. Find the Internet Explorer or Netscape Browser. Then *Apply* the change. Now you can double-click on a GIF file and see it in the Browser.
32. Select the **UP.GIF**, **Click-Right**, select **Rename**, and set the name to **UP\_OLD.GIF**.
33. In the visual composition of the MessageApplication, open the Properties of the first label, select the icon property. Select **File**, browse to the file **dukeMagnify.gif** in the workshop project (IBMVJava\ide\project\_resources\..workshop..project). Close and you have a picture in your application.
34. Now let's replace the splash banner (cow.gif) with our banner. Copy the file **itsobanner.gif** (from **sampcode\ide**) into the workshop project resources directory (IBMVJava\ide\project\_resources\..workshop..project). Use the **Text Search/Replace** facility to change **cow.gif** into **itsobanner.gif**. Rerun the application.  
  
(Note that it would be more appropriate to open the MessageApplicationSplashScreen class and change the cow.gif to the itsobanner.gif.)

### Part IV. Use the New Filter Options

35. In the MessageApplication, go to the **Members** pane.
36. Switch the **Filters** on/off (the icons on the right side) and see what kind of methods/fields are displayed in the list. Change filter options using the context menu -> *Attribute Filters* (in the members pane).

### Part V. Use the New Debugger Functions

37. In the Members pane, set a breakpoint in the method getMessage2.
38. **Run** the application, select *Messages -> Message2* and the **Debugger** window opens at the breakpoint.
39. Select **Window -> Evaluation Area** and **Window -> Visible Variables**. Now we have two new windows. The main debugger window does not show variables any more.
40. Look for the value of the **message2** variable (at the very bottom).
41. Select **Window -> Watches**.
42. Double-click in the **Expression** column, set the expression name to **message2**, and click in the **Value** column to see the current value of the variable.
43. To change the value, type in the **Evaluation** window: message2="Hello", select it, and select **Run** from the context menu.

44. Look again for the value of message2 in the **Visible Variables** and **Watches** window (Use *Refresh* in the Watches window).
45. **Run** the program, and see, how the GUI changes.
46. Select again *Messages* -> *Message2*. When the debugger stops, add this code to the method (before the return):
- ```
int count=0;
while (count<4) {
    message2="Hello world " +count++;
    System.out.println(message2);
}
```
47. Save the code, and set a new breakpoint at the System.out... line, select the breakpoint, select **Modify** (context), set **On Iteration** value to 3, click *OK*.
48. Run the application, and if the debugger comes the 3rd time to this breakpoint, the application is stopped here. Check the value of count in the **Visible Variables** window.
49. Stop the application and close the debugger.

### What you did in this lab

- ☐ Learned, how to create applications with splash screen, menus, menu items, and about boxes automatically with the new SmartGuide for visual applications.
- ☐ Used the new function to create set/get methods for fields
- ☐ Learned, how to use the global search/replace function
- ☐ Used keyword extensions and macros for manual coding.
- ☐ Used the Resource Management function to manage the resources of our project
- ☐ Enable/disable the display of methods/fields with the filters
- ☐ Changed variable values at runtime with the Evaluation Area of the new debugger
- ☐ Enabled the debugger to stop after some iterations of the program



# E2 Accessing Relational Databases with Data Access Beans

## What this Exercise is About

In this lab we want to develop a small application, which retrieves data from the “Sample” database delivered with DB2. We want to use some of the new Data Access Beans from VA JAVA 3.0, and show, how they are to integrate to a simple GUI

## What You Should Be Able To Do

At the end of this lab you should be able to

- ☐ Select data from a DB2
- ☐ Navigate trough result sets
- ☐ Select a single value from the DB

## Introduction

Our application should have the following requirements: One Table, where some columns of the Database are displayed, two buttons to retrieve columns or rows from the resultset, one text field for displaying a selected value of the result set. The data should be retrieved from a local running DB2 database via JDBC.

## Requirements

Make sure that DB2 is running and the “Sample” database is created. (db2sampl.exe).

---

## Exercise Instructions

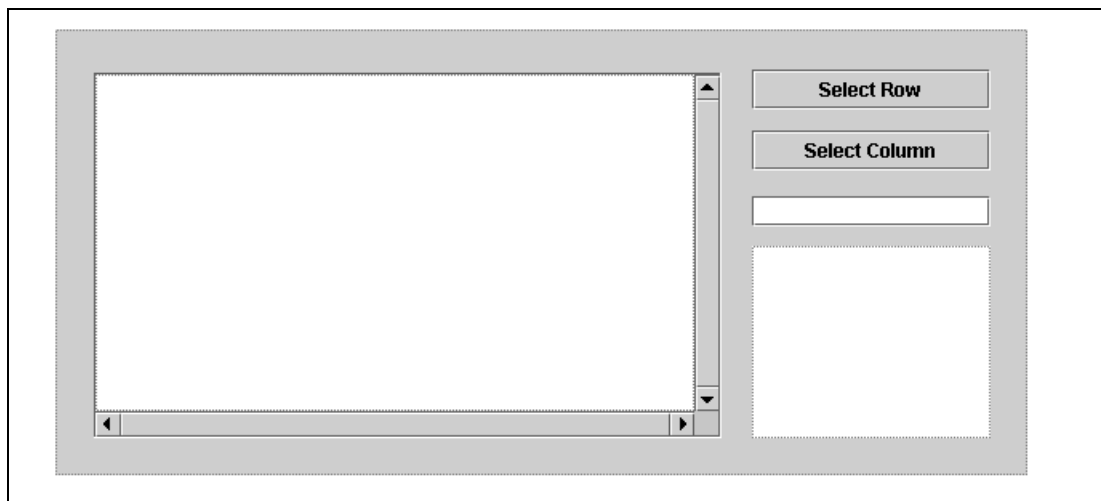
Add the **Data Access Beans** feature to the Workbench.

### Part I: Create a Simple GUI for the application

1. Create a new **itso.vaj3.dabsimple** package to the workshop project.
2. Build a Swing applet named **DABEasySample**, with the following GUI Elements: JTable (displayTable), two JButton (selectRowButton, selectColumnButton), a JList (valueDisplayList) and a JTextField (singleValueField).

Note that you could place the JList into a JScrollPane to allow for scrolling.

3. Your GUI should look as follows



### Part II. Select Data from a Database Table

4. Add a **Select bean** from the Database bean pallet to the free from surface. Open the bean (double-click), set the name to **EasySampleSelect**.
5. Click on **query** to tailor the database connection and select statement.
6. Add a new database access class **EasyDbAccess** (click on *New*) in the package **itso.vaj3.dabsimple**, click OK.
7. **Add** a connection specification. Set the connection name to **easySampleConnection**, the URL to **jdbc:db2:sample**, set the driver choice to **COM.ibm.db2.jdbc.app.DB2Driver**, enable **AutoCommit**, disable **Prompt for logon ID...**, fill out user ID and password (userid,password).
8. Click on Test Connection, if the connection is successful, click *OK*, then *Finish*.
9. Select the **SQL** tab in the properties notebook.
10. Select the database access class **itso.vaj3.EasyDbAccess**.
11. **Add** a new SQL specification, name it **easySampleSQL**, enable **Use SQL Assist...**, click on OK. Wait until the SQL Assist is displayed.
12. Set **statement type** to **Select**, click on the table **Employee**.
13. Click on the **Columns** tab, select the columns **EMPNO**, **FIRSTNAME**, **LASTNAME**, **SALARY**, click on *Add*.



14. Go to the **Sort** tab, add the column **EMPNO**, be sure that **Ascending Order** is selected.
15. Click on the **Mapping** tab to see the SQL data types mapped to Java data types.
16. Select the **SQL** tab to see the generated SQL code. Deselect the **Schema qualified names checkbox**. Click *RUN SQL* to see the result set. Close the result set window, then click on *Finish*, to generate the code. Then click on **OK** and close the properties window of the SampleSelect bean.
17. Now make the Connections:
  - EasySampleSelect, **this** property to the JTable, **model** property
  - FreeFormsurface, **init** event to EasySampleSelect, **execute** method
  - JTable, **selectedRow** property to EasySampleSelect, **currentRow** property. Open the connection and set the source event from none to **mouseReleased**. This sets the current row in the Select bean.
18. Run the program and test.

### Part III. Select a Column or Row from the Result Set

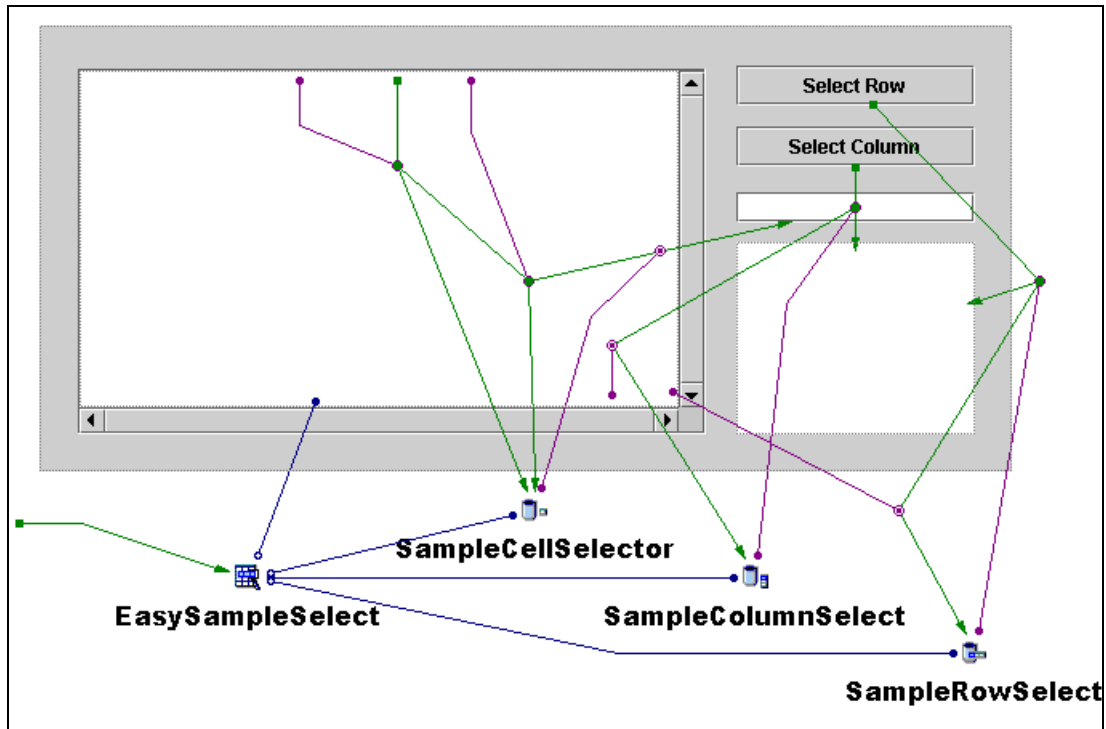
19. Place a **Column Selector** bean from the Database pallet on the free form surface.
20. Open the **properties**, set the bean name to **SampleColumnSelect**.
21. Make the connections.
  - SampleColumnSelect, **model** property to EasySampleSelect, **this**
  - SelectColumnButton, **actionPerformed** to valueDisplayList, **model** and pass the SampleColumnSelect, **this** as parameter (now the JTable knows what to display)
  - **normalResult** (of above) to SampleColumnSelect, **columnNumber** and pass the table, **selectedColumn** as parameter
  - **normalResult** (of above) to valueDisplayList, **repaint** method (to force the painting.... this is not shown in diagrams below)
22. Start the application and test.
23. Implement the same for a selected row using a **RowSelector** bean.
  - Open the properties, set the bean name to **SampleRowSelect**.
  - Make the connections:
    - SampleRowSelect, **model** to EasySampleSelect, **this**
    - Select Row button to valueDisplayList, **model** and pass SampleRowSelect, **this** as parameter (now the JTable knows what to display)
    - **normalResult** to SampleRowSelect, **rowNumber** and pass table, **selectedRow** as parameter
    - **normalResult** (of above) to valueDisplayList, **repaint** method
  - Start the application and test.

### Part IV. Retrieve a Single Value from the Result Set

24. Place a **CellSelector** bean on the free form surface
25. Open the properties, set the bean name to **SampleCellSelector**.
26. Make the connections:
  - SampleCellSelector, **model** property to EasySampleSelect, **this**
  - ScrollPaneTable, **mouseReleased** to SampleCellSelector, **columnNumber**, and pass ScrollPaneTable, **selectedColumn** as parameter

- **normalResult** (of above) to SampleCellSelector, **rowNumber**, and pass ScrollPaneTable, **selectedRow** as parameter
- **normalResult** (of above) to SingleValueTextField, **text**, and pass SampleCellSelector, **String** as parameter (it is an expert property)

27. When finished, your application with all the connections could look as follows:



28. Run the program and test.

## Part V. Optional: Add an Update Action

29. Add a new button (**Update**) to update salary values.
30. Add a **Modify** part from the Database palette, name it **UpdateSalary**.
31. Open the **action** property. Use the same database access class and connection. Switch to the SQL page and click *Add*.
32. Enter **updateSalary** as SQL name and start the SQL Assist SmartGuide.
33. On the Tables page, select an **Update** statement and the Employee table.
34. On the Update page, enter **:salary + 100** as a new value for the SALARY column. On the Condition page enter **:empno** for the EMPNO column.
35. On the SQL page, deselect the **Schema qualified names checkbox** and *Finish*.
36. Connections from the **Update** button actionPerformed:
  - to UpdateSalary, **Parm\_EMPNO\_String** and pass EasySampleSelect, **EMPNO\_String** as parameter
  - to UpdateSalary, **Parm\_SALARY + 100\_String** and pass EasySampleSelect, **SALARY\_String** as parameter
  - to UpdateSalary, **execute** (this updates the row)
  - to EasySampleSelect, **execute** (to refresh the table)
37. Generate and test the applet.

## Part VI. Optional: Add a Stored Procedure Call

38. You must have a JDK installed (for example, JDK 1.1.7) for this exercise.
39. Study the stored procedure: **IncSalaryStp.java** in **C:\Va3Ws\sampcode\spb**. The class has one method(`execute`) with 2 parameters, **increase** and **empno**. The salary of the selected employee is increased by the amount given.
40. Compile the stored procedure. In a Command Window, in **C:\Va3Ws\sampcode\spb**, enter: **javac IncSalaryStp.java**. Copy the resulting **IncSalaryStp.class** file to **C:\SQLLIB\function**.
41. Register the stored procedure. In a **DB2 Command Window**, execute the commands:

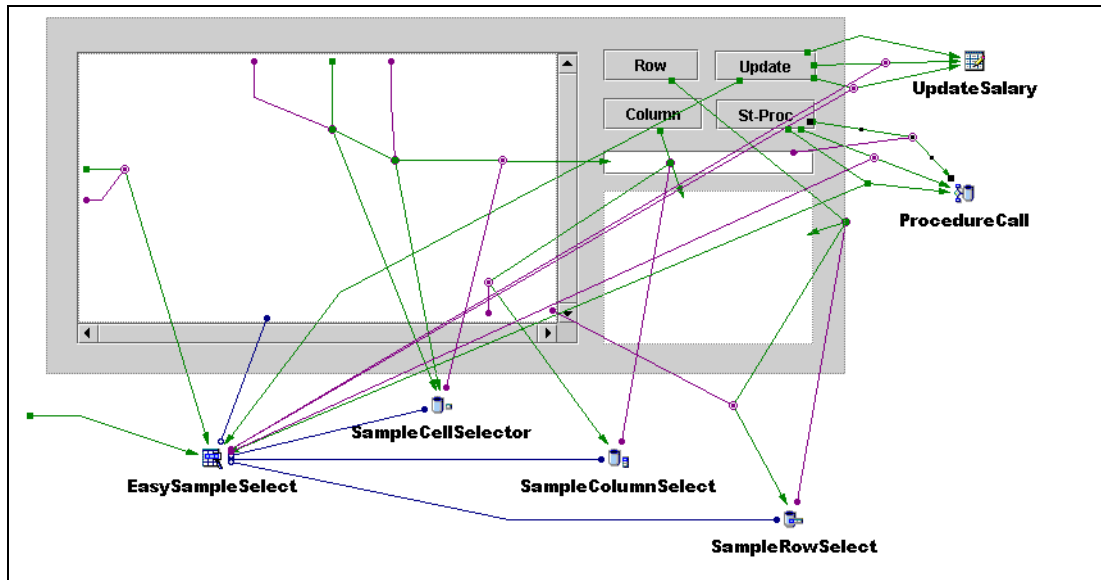
```
db2 connect to sample
```

```
db2 create procedure <userid>.IncSalaryStp (in increase int, in empno char(6)) language  
java parameter style db2general not fenced external name 'IncSalaryStp!execute'
```

Use your logon userid instead of `<userid>`.

42. Configure DB2 to point to the JDK. Open the DB2 Control Center. Select the DB2 instance and *Selected -> Configure*. The last environment entry is **Java Development Kit installation path**. Change the value to the JDK directory, e.g. **C:\jdk1.1.7**. You must stop and restart DB2 for this to take effect:
- ```
db2 force application all  
db2stop  
db2start
```
43. Add a new **St-Proc** button to the application to invoke the stored procedure.
44. Add a **ProcedureCall** bean to the free-form surface. Open the property sheet and open the **procedure** property. Select the same database access class and connection. Switch to the SQL tab and click on *Add*. Name the SQL **increaseSalary** and start the SQLAssist SmartGuide. The **IncSalaryStp** procedure should be listed. Select it and look through the other tabs, then click *Finish* to generate the code.
45. Connections from the **St-Proc** button:
- to **ProcedureCall**, **Parm\_EMPNO\_String** and pass **EasySampleSelect**, **EMPNO\_String** as parameter
  - to **ProcedureCall**, **Parm\_INCREASE\_String** and pass the text field as parameter
  - to **ProcedureCall**, **execute** (this invoked the stored procedure)
  - to **EasySampleSelect**, **execute** (to refresh the table)
46. Generate and test the applet.

The finished application can look like this:



### What you did in this lab

- ☐ Use a Select bean to refresh knowledge about data access beans
- ☐ Used the new data access beans for navigating within a selected row set
- ☐ Specified the SQL code with the SQL Assist Smartguide
- ☐ Connected the properties of the data access beans to a GUI
- ☐ Used the new Modify bean to update data
- ☐ Use the new ProcedureCall bean to invoke a stored procedure

# E3 Servlet Builder with JSP

## What this Exercise is About

In this lab we want to work with the Servlet Builder's new beans to share data and transfer request between two visual servlets and a JSP.

## User requirement

We want to work with a list of employees and give some a raise.

## What You Should Be Able To Do

At the end of this lab you should be able to

- ☐ Create visual servlets
- ☐ Understand an easy way of sharing data between visual servlets.
- ☐ Understand the way of transferring the request between visual servlets
- ☐ Use wrapper beans (variables, servlets, session data)
- ☐ Invoke a JSP using a wrapper bean

## Introduction

We will reuse the data access bean that retrieves employees from the sample table.

---

## Exercise Instructions

Features that must be loaded in the Workbench are:

- ☐ WebSphere test Environment
- ☐ Servlet Builder
- ☐ Data Access Beans

## Part I: Create a Visual Servlet with an Employee Table

1. Create a new **itso.vaj3.servlet** package to the workshop project.
2. Create a new visual servlet (*QuickStart -> Servlet -> Create Visual Servlet*), name it **EmployeeList**, and use the **simple** template. Select the workshop project and the itso.vaj3.servlet package.
3. Lay out the result page as shown below:

The screenshot shows a visual servlet titled "Employee List". It contains a heading "Select Employees for a Raise" in blue text. Below the heading is a button labeled "Retrieve Employees". Underneath the button is a table with four columns: "Number", "Firstname", "Lastname", and "Salary". The table is currently empty. Below the table is a text input field preceded by the label "Signed by:". At the bottom of the layout are two buttons: "Verify with Servlet" and "Execute with JSP".

4. The beans to construct this layout are in the **Servlet** palette middle (text, paragraphs) and bottom (all the form content).
5. Set the title for the page. Use an **Htmltext** for the heading and set `headerLevel=2`. Use a form for the rest:
  - Name the buttons: **retrieveButton**, **verifyButton**, **executeButton**.
  - Name the entry field: **name**
  - The middle part is an **HtmlResultTable**, name it **EmployeeTable**.
  - Drop an **HtmlResultColumn** onto the table and set:
    - bean name: **ColEmpno**
    - columnName: **EMPLOYEE.EMPNO**
    - heading: **Number**
    - selectionType: **Check Box**

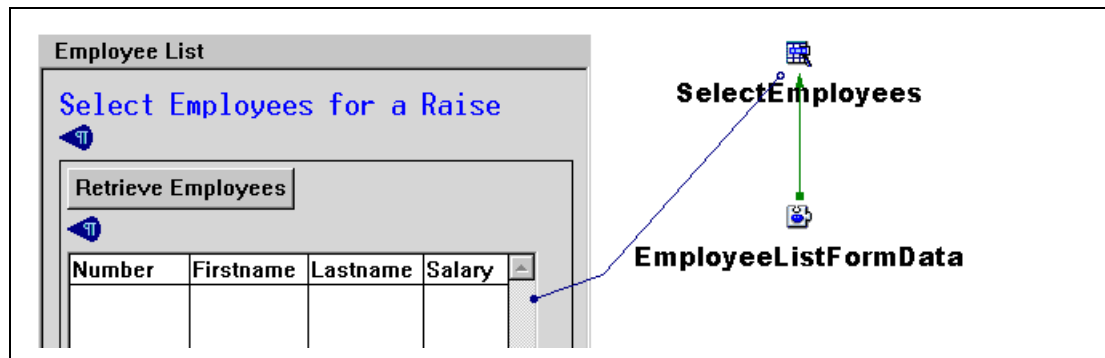
The columnName must match the first column retrieved from the database. The selectionType adds a check box into the output for selection.

  - Optionally drop 3 more result columns and set the columnNames to **EMPLOYEE.FIRSTNAME**, **EMPLOYEE.LASTNAME**, **EMPLOYEE.SALARY**. You can also set the headings, but leave the selectionType as `<none>`.
6. Save the bean to generate the code (*Bean -> Save Bean*). This generates the form data bean (**EmployeeListFormData** class). The form data bean holds all the form input

values for further processing. (Remember: the GUI is the servlet output, not the input, so you cannot connect any GUI data as input values!)

## Part II: Add Logic to Fill the Employees Table

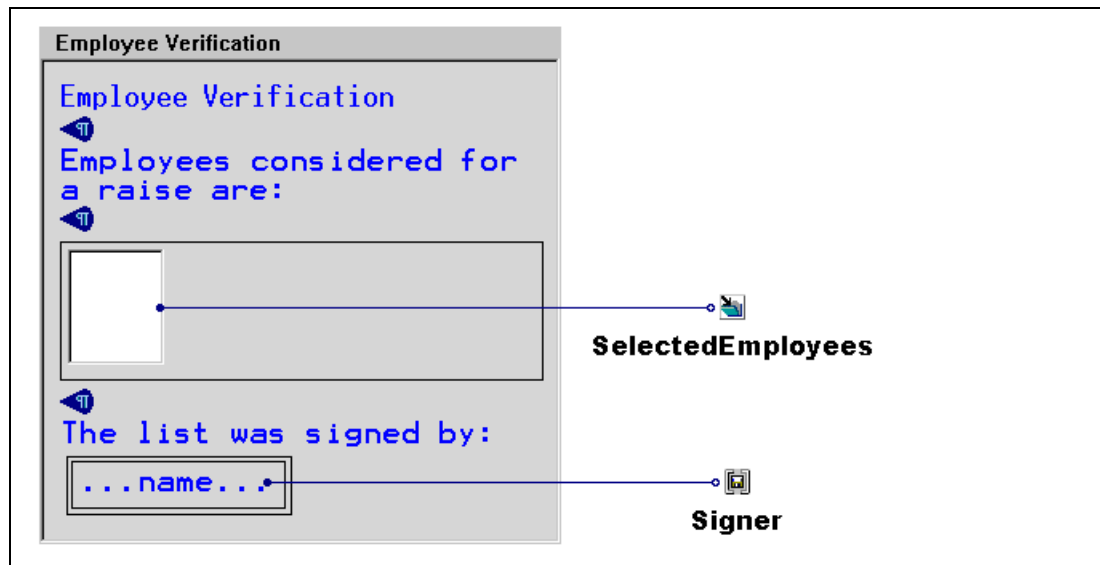
7. Add a **Select** bean (from Database palette) to the free-form surface and set the name to **SelectEmployees**. We want to reuse our query from the data access bean lab. Open the **query** property and select the **itso.vaj3.dabsimple.EasyDbAccess** class: **easySampleConnection**, and **easySampleSQL**.
8. Add a form data bean (Servlet palette, top section) and select the **EmployeeListFormData** class.
9. Connections:
  - SelectEmployees, **this** property to EmployeeTable, **tableModel** property
  - EmployeeListFormData, **retrieveButtonPressed** event to SelectEmployees, **execute** method
10. The composition is shown below:



11. Save the bean to generate the code.
  12. Now let's test what we have:
    - Check the class path of the **SERunner** class: the Servlet Builder libraries must be included (just select all projects).
    - Start the WebSphere Test Environment (*Workspace -> Tools -> Launch WebSphere Test Environment*).
    - Run the servlet by clicking on the *Run* icon. This should start the default Web browser and display the servlet output (be patient). Click on *Retrieve Employees* to get the list.
- If you do not see check boxes in the first column (in front of the employee number) then your *HtmlResultColumn* has the wrong **columnName** property. It has to match what is displayed in your Browser. Fix it....

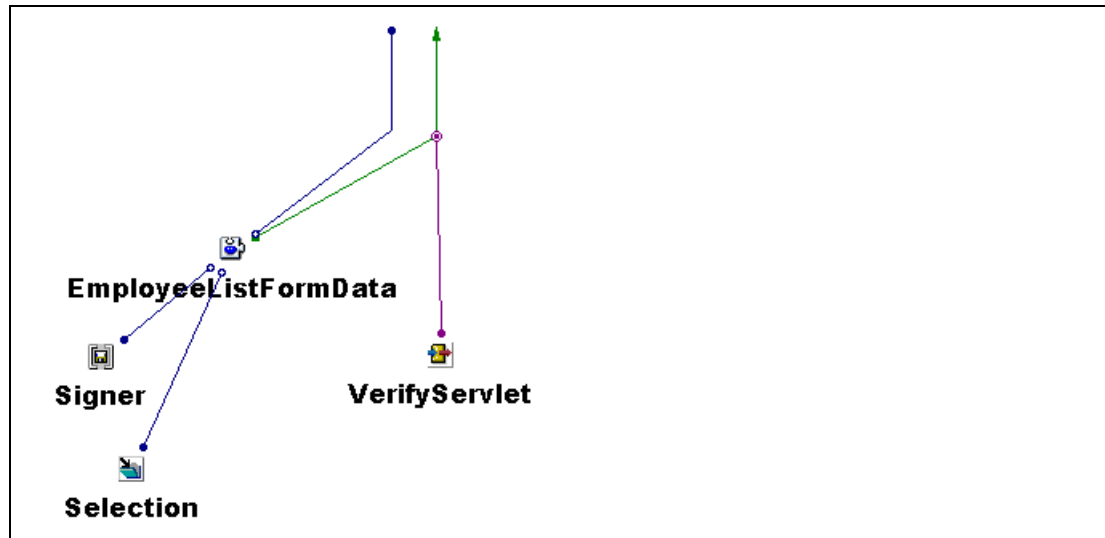
## Part III: Transfer to Another Servlet

13. Instead of developing the second servlet, we just import the code: *File -> Import*, from Directory **sampcode\servlet**, two java files: **EmployeeVerify** and **EmployeeBean**. *EmployeeVerify* is our servlet, *EmployeeBean* we will use later.
14. Open the visual composition of **EmployeeVerify**, it should look as shown below:



15. This servlet displays the selected employees in a list (HtmlList) and the signer as text in a table. You will understand the beans and connections as we develop now the logic on the EmployeeList servlet.
16. Open the **EmployeeList** servlet. Now we add the logic to call the EmployeeVerify servlet when the Verify button is clicked.
17. We will use two beans to store the signer name and the selected employee numbers:
  - Add a **VariableWrapper** bean (Servlet palette top), name it **Signer**, and set the **variableName** property to **signer**.
  - Add an **AttributeWrapper** bean, name it **Selection**, and set the **attributeName** property to **selection**.
  - Connect the **nameString** property of the form data to the **value** property of the Signer bean.
  - Connect the **colEmpnoSelectionItemsStringArray** property of the form data to the **value** property of the Selection bean. Click on *Yes* when warned about unequal types. We convert the String array into an Object.
18. To invoke the EmployeeVerify servlet, we add a bean and a few connections:
  - Add a **ServletWrapper** bean, name it **VerifyServlet**, and open the **servletName** property. Select the **Service Handler** radio button, and find the `itso.vaj3.servlet.EmployeeVerify` servlet in the drop-down list.
  - Connect the **verifyButtonState** property of the form data to the **isTransferring** property of the free-form surface. This indicates that we call another servlet if the Verify button is clicked.
  - Connect the **verifyButtonPressed** event to the **transferToServiceHandler** method of the free-form surface. Pass the **serviceHandler** property of the servlet wrapper bean as value parameter.
19. These steps are illustrated below:





20. Save the bean to generate the code.

21. Now study the VerifyServlet. It uses the same wrapper beans. You have to make sure that the names match (**variableName** and **attributeName** of the beans must be identical in both servlets).

The **value** property of the **selection** bean is connected to the **items** in the HtmlList and the **value** property of the **signer** bean is connected to the **string** of the HtmlText.

22. Test by using your browser to call the servlet again, or start the servlet using the *Run* icon. After selecting a few employees, click on the *Verify* button to invoke the second servlet.

## Part IV: Transfer to a JSP

23. To transfer control to a JSP when the execute button is clicked, we set up a session bean that the JSP can access to get the data.

24. Study the **EmployeeBean** class that we imported earlier. It has two properties: **name** (String) and **selection** (String array). We add the selected employees to this bean, and then add the bean as session data.

25. Add an **EmployeeBean** to the free-form, no tailoring required.

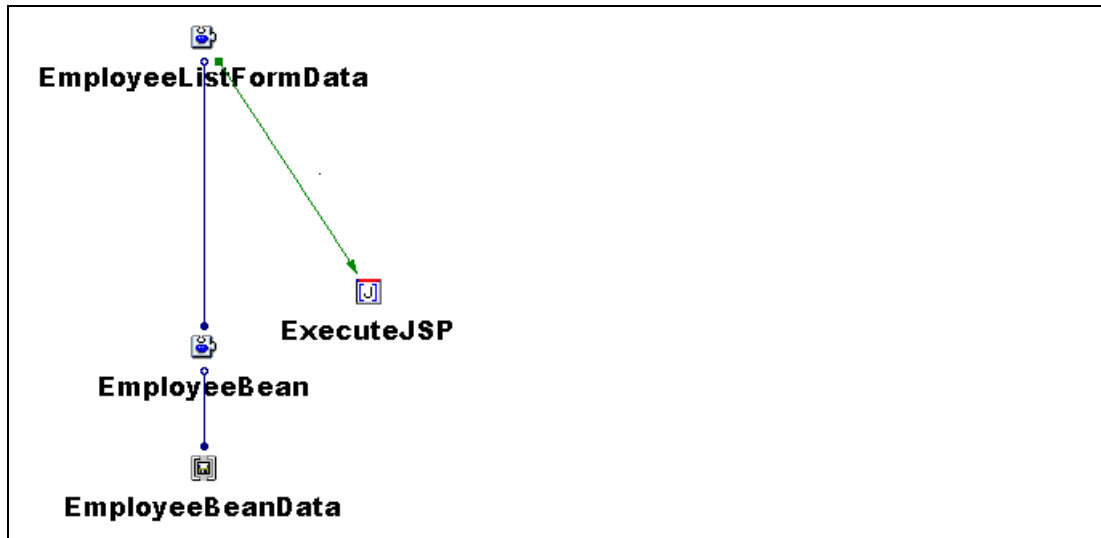
26. Add a **VariableWrapper** bean, name it **EmployeeBeanData** and set the **variableName** property to **employeeBean**. This is the name that the JSP will use.

27. Add a **JSPCallWrapper** bean, name it **ExecuteJSP**, and set the **urlPath** property to **Vaj3Ws/EmpExecute.jsp**.

28. Connections:

- **colEmpnoSelectionItemsStringArray** property of the form data to the **selections** property of the EmployeeBean
- **EmployeeBean**, **this** to **value** of the variable wrapper
- **executeButtonpressed** of the form data to the **forward** methods of the JSP wrapper

29. Connection diagram:



30. The **EmpExecute.jsp** file is in **sampcode\servlet**. This file must be copied to the directory where WebSphere looks for HTML files:

```
d:\IBM\Java\ide\project_resources\IBM WebSphere Test Environment\
hosts\default_host\default_app\web\Vaj3Ws
```

(you may have to create the Vaj3Ws subdirectory).

31. Study the JSP code:

```
<html> <head> <title> Employee Raise JSP </title> </head>
<body>
<h1> Employees with a Salary Raise </h1>
<BEAN name="employeeBean" type="itso.vaj3.servlet.EmployeeBean"> </BEAN>
<table BORDER=3>
  <tr> <b> <td> Number </td> </b>
  <REPEAT index="i">
    <tr> <td> <%= employeeBean.getSelection(i) %> </td>
  </REPEAT>
</table>
<p> Signed by: <%= employeeBean.getName() %>
</body>
</html>
```

The JSP accesses the EmployeeBean with its assigned name (**employeeBean**). Then the get methods (**getSelection(i)** and **getName**) can be called to retrieve the data. The REPEAT tag loops over the selections until the array is exhausted.

32. Save the bean to generate the code. Test by using your browser to call the servlet again, or start the servlet using the *Run* icon. After selecting a few employees, click on the *Execute* button to invoke the JSP.

At this time the JSP is compiled... be patient. It should appear in the Workbench under **JSP Page Compile Generated Code** (project).

33. Check the output. *How did the signer name get there?* We did not connect the signer name to the EmployeeBean in the servlet! (We could have done that.)

The generated code automatically copies any properties in the request block to the JSP bean. The entry field was called **name**, therefore it is copied to the name property of the EmployeeBean!

When finished, you can remove the **Servlet Builder** and the **Data Access Beans** features from the Workbench.

### **What you did in this lab**

- ☐ You created visual servlets and transferred control between servlets and passed data through variable wrappers and session data
- ☐ You invoked a JSP from a servlet and passed data through a bean from the servlet to the JSP
- ☐ You used the WebSphere Test Environment



# E4 XML Parser for Java

## What this Exercise is About

In this lab we want to create an application for interpret our XML file using document type definition (DTD) and a Java parser.

## User requirement

We want to be able of working with drawing data in generic files format, for display in a logical way and use when we need them. This data is base on the lottery drawing internal structure.

- ☐ Drawing number
- ☐ Drawing date
- ☐ Drawing type
- ☐ Drawing Winner Numbers
  - Winner Number
  - Prize

## What You Should Be Able To Do

At the end of this lab you should be able to

- ☐ Create XML files
- ☐ Create DTD files
- ☐ Create a simple parsing for your files
- ☐ Create an applet use that parser
- ☐ Use WebSphere Test Environment to test you parser
- ☐ Basic knowledge in IBM XML Parser for Java packages

## Introduction

We start with define our dialect or document type definition base on the user requirements. Then we create an XML file to be parserd. When we know the file structure, we create an applet to read the file base on a URL. We can also use the Lotus XSL processor to process the XML file into an HTML output.

---

## Exercise Instructions

The **IBM WebSphere Test Environment** feature must be loaded in the Workbench. This also loads the **IBM XML Parser for Java** feature.

### Part I: Create Document Type Definition and XML File

1. We create a DTD file for this exercise. This file describes the dialect used for the XML syntax of our exercise. The name of the file is **winnerNo.dtd** in **sampcode\xml**. The contents of the file is the following:

```
<?xml version="1.0"?>
<!ELEMENT Lottery (DrawingInformation)>
<!ELEMENT DrawingInformation (DrawingNo, DrawingDate, DrawingType, WinnerNo)>
<!ELEMENT DrawingNo (#PCDATA)>
<!ELEMENT DrawingDate (#PCDATA)>
<!ELEMENT DrawingType (#PCDATA)>
<!ELEMENT WinnerNo (InformationNo)>
<!ELEMENT InformationNo (No, PriceTy)>
<!ELEMENT No (#PCDATA)>
<!ELEMENT PriceTy (#PCDATA)>
```

2. We create a sample XML file for this part. The name of the file is **winnerNo.xml** in **sampcode\xml**. The content of the file is the following.

```
<?xml version="1.0">
<!DOCTYPE Lottery SYSEM "winnerNo.dtd">
<Lottery>
  <DrawingInformation>
    <DrawingNo value = "189"></DrawingNo>
    <DrawingDate> May 25,1999 </DrawingDate>
    <DrawingType> Extraordinary </DrawingType>
    <WinnerNo>
      <InformationNo>
        <No> 12345 </No>
        <PriceTy> First Price </PriceTy>
      </InformationNo>
      <InformationNo>
        <No> 25234 </No>
        <PriceTy> Second Price </PriceTy>
      </InformationNo>
    </WinnerNo>
  </DrawingInformation>
</Lottery>
```

3. The two files must be available for testing. Copy the files (**winnerNo.dtd**, **winnerNo.xml**) to the WebSphere Test Environment Resources (d:\IBM\Java\ide\project\_resources\IBM WebSphere Test Environment\hosts\default\_host\default\_app\web\Vaj3Ws). You may have to create the **Vaj3Ws** subdirectory.

4. Start a **Web browser** and see how an XML file is formatted. Microsoft Internet Explorer 5.0 has a formatting routine.

*Decide if you want to build an applet that invokes the XML Parser or if you would rather work with Lotus XSL.*

*Perform Part II for the applet or Part III for Lotus XSL*

## Part II. Create an Applet to Invoke the XML Parser

*You can build this application yourself, or you can build some classes yourself and import the source of other classes from sampcode\xml.*

5. Add a new package to the workshop project and name it **itso.vaj3.xml**.

### Create a Table to display the XML File in a JTable

Import **CreateTable.java** from sampcode\xml into your workshop project and study the code. This is how you could create the class by hand:

6. Our applet needs a model for a JTable. First we will **create the model** for the table. The table has three columns (Level, Name, Value) and many rows from the XML file.
7. Create a **CreateTable** class in **itso.vaj3.xml** that extends from **com.sun.java.swing.table.AbstractTableModel** and import **java.util.\*** package. Let it generate required method skeletons.
8. Add a **private** fields named **rows**, of type **java.util.Vector**, initial value **new Vector()**. Select **Access with getter and setter methods**, but set setter to private. Resulting field should be:

```
private Vector rows = new Vector();
```

9. Add a **private** field named **header**, of type **String**, click on **Array**, initial value **new String[] {"Level","Name","Value"}**, no getter/setter methods. This generates:

```
private String[] header = new String[] {"Level","Name","Value"};
```

10. Change the return statement of **public int getColumnCount()** to the following:

```
return header.length;
```

11. Change the return value of **public int getRowCount()** to the following:

```
return rows == null ? 0: rows.size();
```

12. Change the code of **public Object getValueAt (int arg1,int arg2)** to the following:

```
public Object getValueAt (int arg1, int arg2){
    if (rows == null) return null;
    return((Vector)rows.elementAt(arg1)).elementAt(arg2);
}
```

13. Add a **setValue(Object,int,int)** method with the body:

```
public void setValueAt(Object obj, int arg1, int arg2) {
    ((Vector)rows.elementAt(arg1)).setElementAt(obj, arg2);
}
```

14. Add a **getColumnName(int)** method with the body:

```
public String getColumnName(int column) {
    return header[column];
}
```

15. Add an **addValue(Vector )** method:

```
public void addValue(Vector v) {
    rows.addElement(v);
}
```

16. Add a **changeLastValue(String)** method:

```
public void changeLastValue(String str) {
    String oldValue = (String) ((Vector)rows.elementAt( rows.size()-1 )).elementAt(2);
    if (oldValue.equals("")) setValueAt(str, rows.size()-1, 2);
    else setValueAt(oldValue+" "+str, rows.size()-1, 2);
}
```

## Create a Document Handler Class for the XML file

Import **DrawingHandler.java** from sampcode\xml into your workshop project and study the code. This is how you could create the class by hand:

17. Select the **itso.vaj3.xml** package and add a class to the package (*Class* icon, or *Add -> Class*). Name the class **DrawingHandler**, as a subclass of **org.xml.sax.HandlerBase**. On the next page, import **org.xml.sax.\*** and **java.util.\*** and click *Finish*.

18. Add a **private** field named **table**, of type **CreateTable**, initial value **new CreateTable()**, select **Access with getter and setter methods**, but set setter to private (it is not used).

19. Add a **private** field named **count**, of type **int**, initial value **0**, no getter/setter methods (private int count = 0;).

20. Add a **public void** method with **two parameters** and name it **startElement**. One parameter is **java.lang.String** and call it **name**, the other is **org.xml.sax.AttributeList** and call it **atts**. Add the following code to the method.

```
public void startElement (String name, AttributeList atts){
    String value = atts.getValue(0);
    Vector temp = new Vector();
    temp.addElement(" " + count);
    temp.addElement(name);
    if (value != null) temp.addElement(value);
    else temp.addElement("");
    getTable().addValue(temp);
    count++;
}
```

This method is the key: it is called from the Parser for every element in the XML file. We add the element as a Vector to the rows table and increase the level (count).

21. Add a **public void** method with three parameters and name it **characters**. The parameters are a char array and two int. Here is the code:

```
public void characters(char [] ch, int start, int lg) {
    try {
        String data = (new String(ch,start,lg)).trim();
        if (!data.equals("")) getTable().changeLastValue(data);
    }
    catch (Exception e) {}
}
```

This method is called with the text between the start and end tags. We update the last value in the table with the new data.



22. Add a **public void** method with one parameter (String) and name it **endElement**. Here is the code:

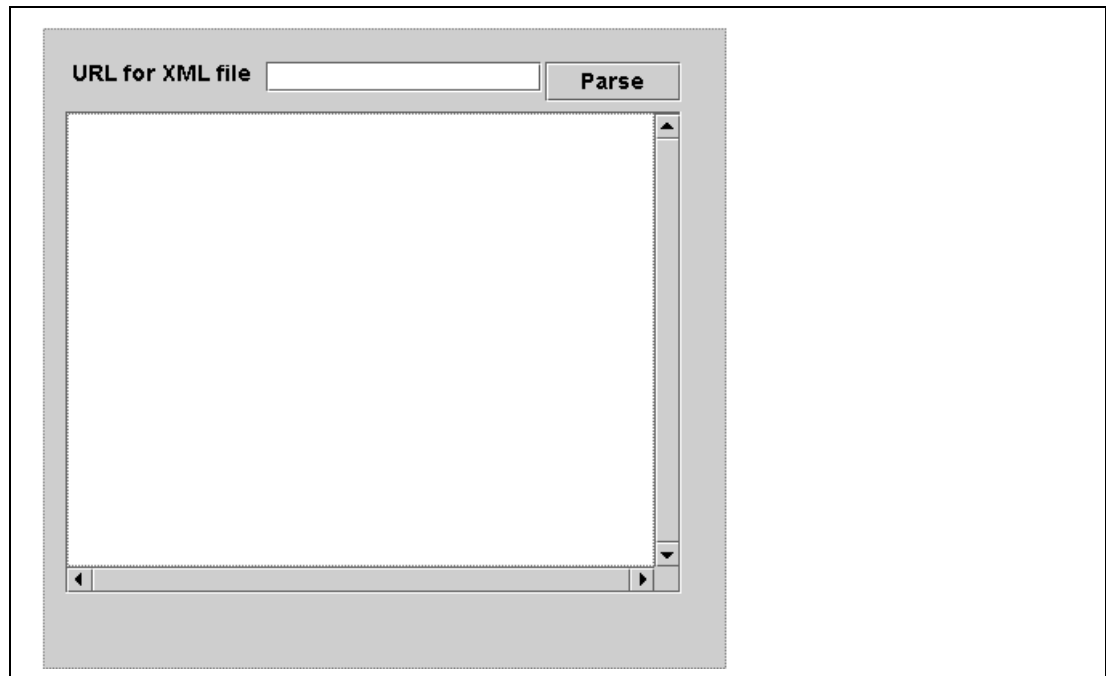
```
public void endElement(String name) {  
    Vector temp = new Vector();  
    count--;  
    temp.addElement("" + count);  
    temp.addElement("/"+name);  
    temp.addElement("");  
    getTable().addValue(temp);  
}
```

This method is called for the end tag. We decrease the counter and add the element to the table.

### Create a Swing Applet to Invoke the Parser

Import **DrawingApplet.java** from `sampcode\xml` into your workshop project and study the visual composition. This is how you could create the class by hand:

23. Create a new JApplet named **DrawingApplet**. Compose visually as shown.

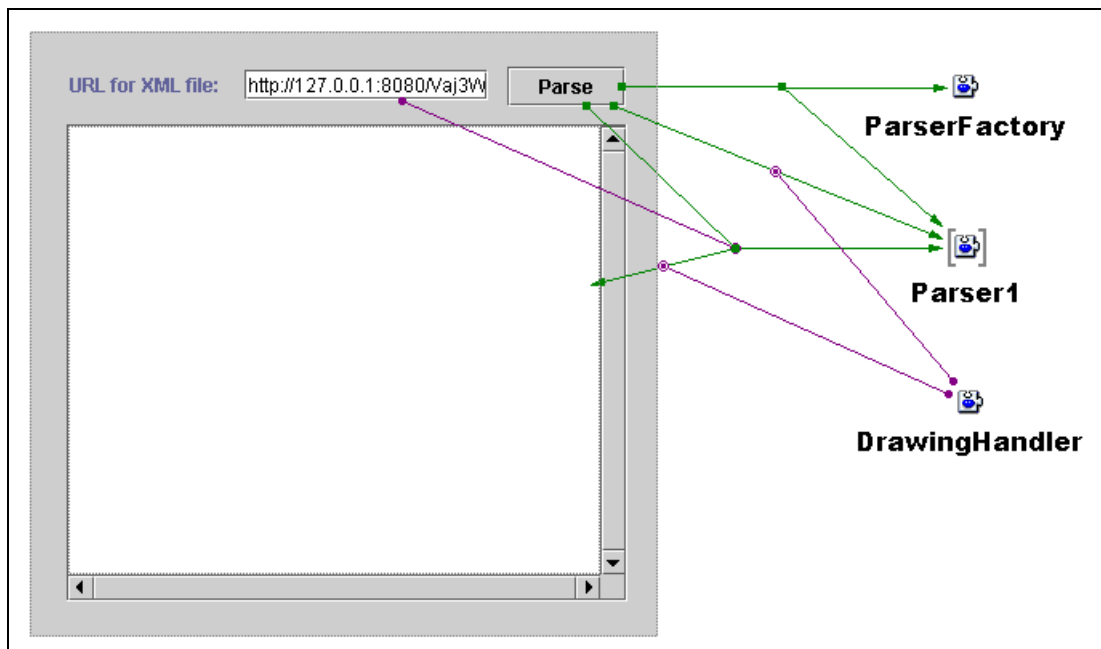


- Add a label: **URL for XML file**
- Add a text field **xmlURLValue**, with a default value of **http://127.0.0.1:8080/Vaj3Ws/winnerNo.xml** (this points to the XML file so we do not have to enter the long string).
- Add a button (**parseButton**) with the text **Parse**
- Add a JTable for the result display

24. Add 3 beans to the free-form surface:

- **org.xml.sax.helpers.ParserFactory** as a **class**
- **org.xml.sax.Parser** as a **variable**
- **itso.vaj3.labxml.DrawingHandler** as a **class**

25. Connect the **parserButton** to the **makeParser(java.lang.String)** method of the **ParserFactory** and set the connection parameter to the string **com.ibm.xml.parsers.ValidatingSAXParser**. Connect the **normalResult** to the **this** property of the **Parser** variable. Now we have a parser.
26. Connect the **parserButton** to the **setDocumentHandler(...)** method of the **Parser** bean and pass the **this** property of the **DrawingHandler** bean as parameter. The parser knows now which class to call for every tag in the XML file.
27. Connect the **parserButton** to the **parse(String)** method of the **Parser** bean and pass the text of the **xmlURLValue** bean as parameter. This starts the parsing process.
  - Connect the **normalResult** event to the **model** property of the **JTable** and pass the **table** property of the **DrawingHandler** as parameter (to draw the table).
28. The completed applet is shown below. Save it to generate the code.



## Running the Applet

29. **Compute the class path** of the applet (*Bean -> Run -> Check class path*) and click *Compute Now*. Before executing the applet, launch the **WebSphere Test Environment** (*Workspace->Tools->Launch WebSphere Test Environment*). Make sure the class path of SERunner includes all projects.
30. Run the applet. Enter the URL: **http://localhost:8080/Vaj3Ws/winnerNo.xml** and click the *Parse* button.

## Optional Enhancement: Remove Unnecessary End Tags

31. Another extension would be to eliminate the end tags for the values. You can save the tag name and if the **endElement** method is called with the same name, do not add it to the table:
  - create a private field **String oldName = null;**
  - add to the **startElement** method: **oldName = name;**
  - change the last line in the **endElement** method to read:
 

```
if (!oldName.equals(name)) getTable().addValue(temp);
```

## Optional Enhancement: Display XML as Tree

32. The Swing **JTree** is very suitable to render the XML file graphically.

33. Add a **JScrollPane** and a **JTree** inside the scroll pane at the bottom of the GUI.

34. Enhance the **DrawingHandler** class to fill the tree with the node and values.

35. Add to the class definition:

```
import com.sun.java.swing.*;
import com.sun.java.swing.tree.*;

// JTree support
JTree    drawingTree = null;
TreeModel drawingTreeModel = null;
Vector    drawingTreeNode = new Vector();
```

36. Add one method:

```
public TreeModel getDrawingTreeModel() {
    return drawingTreeModel;
}
```

37. Change the **startElement** method to manage the tree (before count++):

```
// JTree
DefaultMutableTreeNode aNode;
if (value != null)
    aNode = new DefaultMutableTreeNode(name+"="+value);
else
    aNode = new DefaultMutableTreeNode(name);
if (count != 0) {
    DefaultMutableTreeNode parent =
        (DefaultMutableTreeNode) drawingTreeNode.elementAt(count-1);
    parent.add(aNode);
}
drawingTreeNode.addElement(aNode);
count++;
```

38. Change the **endElement** method to manage the tree and set up the model at the end:

```
// JTree
if (count == 0) {
    drawingTree = new JTree(
        (DefaultMutableTreeNode) drawingTreeNode.elementAt(0) );
    drawingTreeModel = drawingTree.getModel();
}
drawingTreeNode.removeElementAt(count);
```

39. In the **DrawingApplet** add one connection:

- Connect the **normalResult** event of the **parse** method call to the **model** property of the **JTree** and pass the **drawingTreeModel** property of the **DrawingHandler** as parameter (to draw the tree).

## Part III: Lotus XSL Processor

40. Create a new project in the Workbench and name it **Lotus XSL**.

41. We must import the Lotus XSL classes. Choose one of two ways:

- Import the whole project from a provided Repository file (**sampcode\xml\lotusxsl.dat**). Note that after a repository import you have to add the `com.lotus.*` packages to the Lotus XSL project in the Workbench.
- Import the source files (.java) to the project from a provided Jar file (**sampcode\xml\lotusxsl.jar**).

Now you should have a few **com.lotus.xsl.\*** packages in the Lotus XSL project.

42. Study the XSL translator file **winnerNo.xsl**:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">
<xsl:strip-space elements="Lottery DrawingInformation WinnerNo"/>
<xsl:template match="Lottery">
  <html>
    <head> <title> Winner Numbers </title> </head>
    <body>
      <h1> Winner Numbers </h1>
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>
<xsl:template match="DrawingInformation/DrawingNo">
  <h2> Drawing Number is <xsl:value-of select="@value"/> </h2>
  <xsl:apply-templates/>
</xsl:template>
<xsl:template match="DrawingInformation/DrawingDate">
  <ul> <li> Drawing Date is <b> <xsl:apply-templates/> </b> </li> </ul>
</xsl:template>
<xsl:template match="DrawingInformation/DrawingType">
  <ul> <li> Drawing Type is <b> <xsl:apply-templates/> </b> </li> </ul>
</xsl:template>
<xsl:template match="WinnerNo">
  <div>
    <table border = "1">
      <tr> <td> <b> Winner Number </b> </td>
        <td> <b> Price </b> </td> </tr>
      <xsl:apply-templates/>
    </table>
  </div>
</xsl:template>
<xsl:template match="WinnerNo/InformationNo">
  <tr>
    <xsl:apply-templates/>
  </tr>
</xsl:template>
<xsl:template match="WinnerNo/InformationNo/No">
  <td> <xsl:apply-templates/> </td>
</xsl:template>
<xsl:template match="WinnerNo/InformationNo/PriceTy">
  <td> <xsl:apply-templates/> </td>
</xsl:template>
</xsl:stylesheet>
```

For each XML tag there is a template for the resulting HTML tags.

43. First we can use the **com.lotus.xml.Process** class to run the XSL processor. Find the class and open its properties (*Selected -> Properties*). On the **Program** pane, enter the command line parameters:

```
-v -xsl Vaj3Ws\winnerNo.xsl -in Vaj3Ws\winnerNoX.xml -out Vaj3Ws\winnerNo.html
```

We are using the **winnerNo.xsl** file to translate the **winnerNoX.xml** file into a **winnerNo.html** file. (Note: **winnerNoX.xml** is the same as **winnerNo.xml** except that there is no reference to the **winnerNo.dtd** file.)

44. To work with these file, copy **winnerNo.xsl** and **winnerNoX.xml** into the resources directory **d:\IBM\Java\ide\project\_resources\Lotus XSL\Vaj3Ws**. (You have to create the **Vaj3Ws** subdirectory.)
45. Before running the **Process** class, check the class path (*Selected -> Run -> Check Class Path*) and use the *Compute Now* button (the XML Parser project will be added.)
46. Now run the **Process** class and watch the output messages in the Console. Then check the **Lotus XSL\Vaj3Ws** subdirectory and the **winnerNo.html** output file should be there.
47. Now we want to invoke the XSL processor from a servlet. Suppose that the servlet gets an XML file from a back-end system (database, transaction) and must display the file nicely in a browser.
48. The servlet is invoked from an HTML file. Copy the **winnerXSL.html** file and also **winnerNo.xsl** and **winnerNoX.xml** (in **sampcode\xml**) into the WebSphere Test Environment Resources directory **d:\IBM\Java\ide\project\_resources\IBM WebSphere Test Environment\hosts\default\_host\default\_app\web\Vaj3Ws**.
49. Import the servlet source code (**sampcode\xml\XslServlet.java**) into your workshop project. Study the **doPost** method. It allocates an XSL processor and calls it with the parameters passed from the HTML form. The output **PrintWriter** is passed to the XSL processor so that the resulting HTML file is visible as output in the browser.
50. Configure the WebSphere Test Environment **SERunner** class and make sure that the class path include all the projects (Lotus XSL, workshop, XML Parser, etc.). Then start the WebSphere Test Environment (*Tools -> Launch WebSphere...*).
51. In a browser enter **http://127.0.0.1:8080/Vaj3Ws/winnerXSL.html** to display the HTML file. Submit the form and run the servlet. You should see the output in the browser.

### What you did in this lab

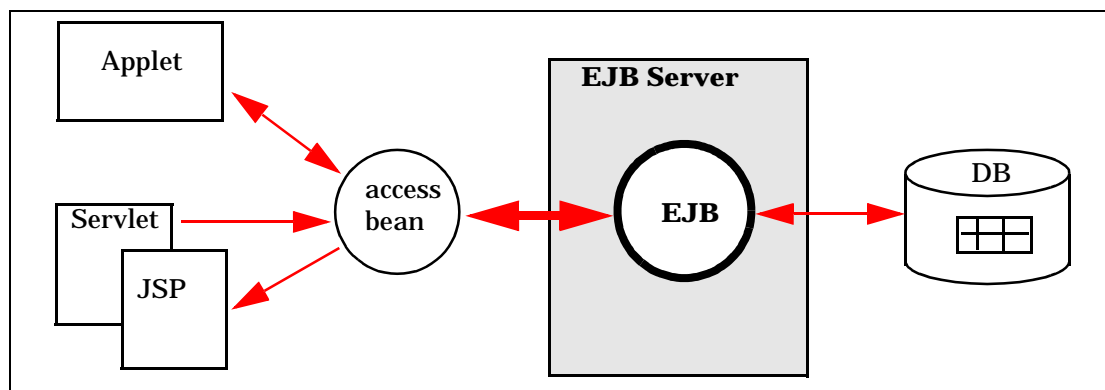
- ☐ Defined a new markup languages
- ☐ Created DTD and XML files
- ☐ Created a simple parsing class for your files
- ☐ Created an application to drive the parser
- ☐ Used WebSphere Test Environment to test you parser
- ☐ Acquired basic knowledge in IBM XML Parser for Java packages
- ☐ Optionally learned something about the Swing tree
- ☐ Optionally used the Lotus XSL processor to translate XML into HTML



# E5 Developing and Using an EJB

## What this Exercise is About

In this lab we create an EJB that maps to a table in the DB2 sample database. We test the EJB and then we write two applications, a GUI and a servlet that uses the EJB through an access bean.



## What You Should Be Able To Do

At the end of this lab you should be able to

- ☐ Create an EJB and map it to a database table
- ☐ Run the Test Client for the EJB
- ☐ Create an access bean for the EJB
- ☐ Create an applet and a servlet/JSP that uses the EJB through the access bean

## Introduction

We create an EJB for the department table of the DB2 sample database.

---

## Exercise Instructions

Before starting with this exercise, load the **IBM EJB Development Environment** feature.

### Part I: Create a Container-Managed Entity Bean

1. Create a new **itso.vaj3.ejb** package in the workshop project.
2. In the Workbench, switch to the **EJB** pane.

#### Define an EJB

3. Use the context menu in the **Enterprise Beans** pane, or the EJB menu, and select *Add -> EJB Group*. Select the workshop project, and give the new group the name **ITSO\_EJBs**.
4. Now we want to create a **container-managed persistent enterprise bean**. From the context menu, select *Add -> Enterprise Bean*. In the SmartGuide enter **Department** as the bean name, **container-managed persistence** as the bean type, create a new bean class, the workshop project (pre-filled), the **itso.vaj3.ejb** package, no superclass. Click *Next*.
5. Leave the defaults for home interface, remote interface, and key class. Click on *Add* for new CMP fields. In the **Create CMP Field SmartGuide** enter **deptno** as name, **java.lang.String** as type, and select **Key Field**. Click *Finish*.
6. Add additional CMP non-key fields, and select both **Access with getter and setter methods**, and **Promote getter and setter methods to remote interface** (public methods). The additional fields are: **deptname**, **mgrno**, **admrdept**, **location**, all String. This matches the department table of the DB2 sample database.
7. Click *Finish* when all fields are defined. The EJB window shows now the types (Department, DepartmentBean, DepartmentBeanFinderHelper, DepartmentHome, and DepartmentKey) and the members of each type.
8. Switch the middle pane from Types to **Properties** (click on the **F** icon). Select a field and check that its context menu has **Container Managed** checked. (You can also see this from the DB icon after the field name.) The **deptno** field also shows the **key icon**. Click the **C-I** icon to get back to the **Types** pane.
9. Now we have to define the table and mapping for the EJB. Select *EJB -> Open to -> Database Schemas*. This opens the **Schema Browser** of the Persistence Builder.

#### Map the EJB to a Database Table

10. Select *Schemas -> Import / Export Schemas -> Import Schema from Database*. Enter **ITSO\_EJBs** as schema name. In the **Database Connection Info** window select the **app.DB2Driver**, the **jdbc:db2:sample** data source, **userid**, and **password**. Click on *Test* to make sure the connection works. Click *OK*.
11. In the **Select Tables** dialog, find the **DEPARTMENT** table (*Build Table List* button) and click on *OK*.
12. To do the mapping, select *EJB -> Open to -> Schema Maps*. This opens the **Map Browser** of the Persistence Builder.
13. From the Datastore\_Maps menu, select **New EJB Group Map** (not Datastore Map!). Enter **ITSO\_EJBs** as name, EJB Group, and Schema. Click *OK*.



14. Select the new map and its only persistent class. Select *Table Maps -> New Table Map -> Add Table Map with No Inheritance*. Select the DEPARTMENT table in the dialog.
15. Select the new map and *Edit Property Maps* in the context menu. Map each attribute as **simple** type to the matching column (you have to select each column) of the table. Click *OK*.
16. Save the schema (*Schema -> Save Schema*) and the map (*Data\_store -> Save*) in a new **itso.vaj3.ejb.metadata** package.

## Generate the Code

17. Now we can generate the underlying code for our enterprise bean. In the **Workbench**, select the Department bean and *Generate -> Deployed Code* from the context menu. Afterwards select *Generate -> Test Client*. As a result a number of classes are generated into the **itso.vaj3.ejb** package.
18. We can see the generated types in the **EJB Types pane** by clicking on the “list” icon.

## Configure and Start Servers

19. Select the **ITSO\_EJBs** group and *Add To -> Server Configuration* from the context menu. This opens the **EJB Server Configuration** window.
20. Start the **Location Service Daemon** (select the server and click the *Start* icon).
21. Select the **Persistent Name Server** and open its **Properties**. Specify **InstantDB** as data source. This is a simulated database, no JDBC driver is required.  
  
You can also specify a real database (**jdbc:db2:sample**) and use the **app.DB2Driver** as connection type, with a userid and password.  
  
Close the properties dialog and then start the Persistent Name Server. Watch the Console window (until *Server open for business*).
22. Open the properties of the **EJB Server** and specify the same values as data source (**jdbc:db2:sample**), connection type (**app.DB2Driver**), userid, password. Start the EJB Server. Watch the Console window (until *Server open for business*).

## Test Client

23. In the **EJB Server Configuration** window, select the **Department** bean and *Run Test Client* (or use the yellow running man icon).
24. The **Test Client** opens for the Department bean. Click on *Connect*. Watch the messages: An instance of the EJB home was found.
25. Let's find a department bean. Select **findByPrimaryKey**. Click on *New* in the Parameters pane. In the **DepartmentKey** dialog, select **newDepartmentKey(String)** and enter **E21** as the value. Click *Send* to create a department key object. Click *Done*.
26. Click *Send* in the Test Client. The department object is retrieved and the window switches to the **remote interface**. Now you can look at attribute values and update the bean.
27. Select the **getDeptname** method and click *Send*. The result SOFTWARE SUPPORT is displayed. The method **getAdmrdept** displays E01, **getMgrno** displays 000100, **getLocation** displays null.
28. Select **setLocation(String)**, enter **London** as parameter, and click *Send*. This changes the table in the database. Open a **DB2 Command Window** and verify the change (connect to sample, select \* from department).

29. To retrieve another department switch back to the **home interface** and find another bean (A00, E01, etc).
30. To create a department use the **create(String)** method. Enter **X66** as the parameter and click *Send*. You will get a DB2 error message about NULL columns not being allowed. The table requires that deptname and admrdept are not null.  
  
**Note:** Because of a bug in the EJB container code no error occurs. The values from the last department bean are used. This will be fixed....
31. In the **Workbench**, EJB window, select the Department bean and its **ejbCreate** method. Add two statements to initialize the required fields:
 

```
deptno = argDeptno;
deptname = "Unknown";
admrdept = " ";
```
32. Try the create again in the **Test Client** and it works. Check the row in the DB2 Command Window. Update the values of the columns using the setter methods (deptname = ITSO, admrdept = D01, mgrno = 000070, location = San Jose).
33. Stop the Test Client (close). Stop the 3 servers from the Server Configuration window and close the window.

### Prepare Deployment

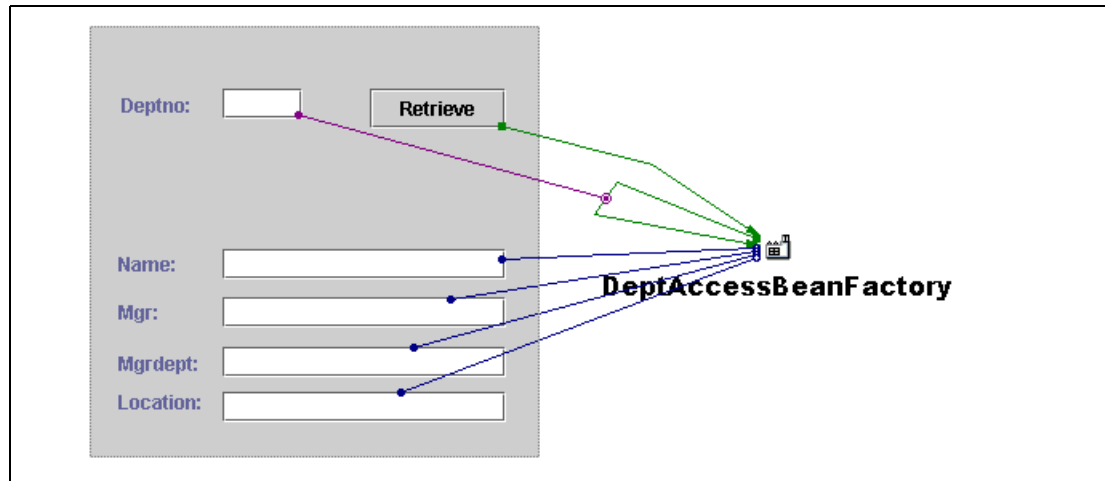
34. To deploy an EJB to WebSphere we create a JAR file. We can export a deployed version of the bean or we can let WebSphere do the work. Because we already have a database mapping export the deployed version.
35. Select the **Department** EJB and *Export -> Deployed JAR*. We can write the jar file directly to a WebSphere directory, but for now we just create the file in any directory and name it **DepartmentEJS.jar**. (If we select *Export -> EJB JAR*, then WebSphere will deploy the bean and recreate the mapping.)

## Part II. Create an Access Bean

36. Now let's create an access bean for the **Department** bean. Select *Add -> Access Bean* from the context menu.... be patient, it takes a while for the SmartGuide to start.
37. Select **Java Bean Wrapper** and in the next panel **findByPrimaryKey** for zero argument constructor. Click *Finish*. A class with the name **DepartmentAccessBean** is created.
38. Study the methods of the access bean to understand how it communicates with the EJB.

## Part III. Use the Access Bean in a GUI

39. Construct a JApplet that uses the access bean. Create a **DepartmentTest** class in the new **itso.vaj3.ejb.gui** package.
40. Create the GUI layout as shown below.



41. Drop a **factory** on the free-form surface and change the **type** to **DepartmentAccessBean**.

42. Connections:

- Connect the **Retrieve** button to the factory constructor **DepartmentAccessBean()**.
- Connect the **this** event of factory to **setInitKey\_deptno** method (of factory) and pass the **deptno** entry field text as parameter. The EJB is now prepared.
- Connect the four attributes of the factory to the matching text fields. Select **this** as the source event, **none** as target event.

When the bean is constructed, the this event forces the attributes to be propagated to the GUI. The first method call finds the real EJB!

43. **Save** the bean to generated the code.

44. Set the **class path** for the GUI (*Bean -> Run -> Check Class Path*) and click on *Compute Now* to get the EJB libraries added.

45. Start the 3 EJB servers from the Server Configuration window (*EJB -> Open To -> Server Configuration*).

46. Test the applet and enter **A00** as department number. Be patient ...

47. If it does not work, check the sequence of the connection. The factory object must be set first, then the deptno passed as key, then the attributes retrieved. Open Reorder connections from and make sure that setInitKey\_deptno is on the top.

## Part IV. Use the Access Bean in a JSP and Servlet

48. Browse the **sampcode\ejb** directory. You find source code, HTML, and JSP.

49. Some of these files must be placed into proper VA Java directories:

- JSP and HTML files must be copied to  
**d:\IBMVJava\ide\project\_resources\IBM WebSphere Test Environment\hosts\default\_host\default\_app\web\Vaj3Ws**  
(you may have to create the **Vaj3Ws** subdirectory)
- Servlet configuration files (.servlet) must be copied to  
**d:\IBMVJava\ide\project\_resources\...workshop...\itso\vaj3\ejb\servlet**  
(you have to create the **itso\vaj3\ejb\servlet** subdirectories)

50. In the **Workbench**, select the workshop project and import the two Java source files (*File -> Import*, from directory). This creates an **itso.vaj3.ejb.servlet** directory. Let's not study these classes yet.
51. Study the **OneDept.jsp** source. This JSP runs by itself, without a servlet. It allocates the **DepartmentAccessBean** and retrieves one department into local variables. The information is displayed in a table.
- Because the access bean throws exception, a try/catch block must be used. This coding is not very nice.
52. Run this JSP. The EJB servers must be running. Start the **WebSphere Test Environment** (make sure the class path of the SERunner class includes all the projects). Start a browser and enter **http://127.0.0.1:8080/Vaj3Ws/OneDept.jsp**. This should compile the JSP and run it.
53. In a second example we want to use a servlet that sets up a bean with the data and then calls a JSP.
- **EJBDept.html** contains a form to invoke the servlet and pass a department number.
  - **EJBDeptBean.java** is a wrapper bean that instantiates the department access bean and retrieves the attributes into local variables in the execute method. Getters are provided to get the attribute values. One setter method must be called first to set the department number.
  - **EJBDeptServlet.java** is the servlet. In the **performTask** method, the EJBDeptBean is allocated and registered. The department number is retrieved from the HTML form and passed to the bean. Then the execute method is called. Finally the AnyDept.JSP is invoked for the output.
  - **AnyDept.jsp** is the JSP that produces the output. Notice how much simpler the coding is because we are using a wrapper bean that handles the exceptions.
54. Run this example by entering **http://127.0.0.1:8080/Vaj3Ws/EJBDept.html** in the browser.

## Part V. Optional Extension

55. Extend the applet for inserts and/or updates. Define new push buttons, Update and Insert.
- For update (after retrieving the EJB), pass the changed attribute values to the access bean. This invokes the setters on the EJB.
  - For insert call the constructor with a key string. This creates a new EJB. Then pass the entry fields to the access bean attributes using the Update button.

### What you did in this lab

- ☐ You creates a container-managed EJB
- ☐ You mapped the EJB to an existing table
- ☐ You created an access bean for the EJB
- ☐ You used the access from an applet and from a servlet/JSP

# E6 Using the HPJ Compiler and the Distributed Debugger

## What this Exercise is About

In this lab we want to change an existing Java application running in a JVM to a native (x86) code running only on Windows.

Later, we will debug both of this applications with the distributed debugger.

## What You Should Be Able To Do

At the end of this lab you should be able to

- ☐ Use the HPJ to create faster Java applications
- ☐ Use the debugger for stepping through the code, inspecting data, and changing values at runtime

## Introduction

We starting with a little java program, we will compile these program inside and outside VA Java with the HPJ, and then debug the two programs with the remote debugger.

## Requirements

Be sure, the **EnterpriseToolkit for Workstation** is installed as well the **Distributed Debugger**. We also need a JDK installed on your local machine. We will use the Hanoi sample from the IBM Java Examples.

---

## Exercise Instructions

Add the **com.ibm.ivj.examples.hanoi** package to your workshop project (this is faster than adding the whole examples project).

### Part I: Compile a Java Program with HPJ Inside VA Java

1. Go to the Workbench, select the **com.ibm.ivj.examples.hanoi** package, and select *Tools -> ET/Workstation -> Properties* (be patient...).
2. Click on the icon before **File export choices** to allow changing the properties.
3. Select **Export both .class and .java files**.
4. Set the **Export Directory** to **c:\Vaj3Ws\sampcode\hpj** (where the workshop sample code is).
5. Click on the (+) to expand **Compile Session**.
6. In Basic Options, select **Build an executable for main class**, *Browse* for the Class **Hanoi**, in the package **com.ibm.ivj.examples.hanoi**, Click **OK**. Set the **Target Name** to **hanoi.exe**.
7. Select **Generate Debug Information**.
8. Under **Optimization Options**, select **Optimize**.
9. Click on the left side on **Run/Debug Session**. Set **Specify main class** as **com.ibm.ivj.examples.hanoi.Hanoi**, or **Browse** for it.
10. Enable the **Classpath**, click on **OK** to close the dialog.
11. Select the **com.ibm.ivj.examples.hanoi** package, select *Tools -> ET/Workstation -> Compile* (be patient...).
12. Use the command line of Windows, go to the directory **c:\Vaj3Ws\sampcode\hpj** and start **hanoi.exe**.

### Part II. Compile a Java Program with HPJ Outside VA Java

13. From the Workbench, export the package **com.ibm.ivj.examples.hanoi** to the directory **c:\Vaj3Ws\sampcode\hpj**. Export as class and java files, and include the debug informations in the .class files.

Actually all files are already there from the previous step.

14. Go to the command line and change to the directory **c:\Vaj3Ws\sampcode\hpj**.
15. Enter on the command line:

```
hpj -exe -g -make -o p2hanoi com\ibm\ivj\examples\hanoi\Hanoi.java
```

16. Execute **p2hanoi.exe**

### Part III. Debug a Compiled Java Program

17. Go to a command line and start the debugger with the command **idebug**.
18. In the Load Program Dialog, go to the **Compiled (Intel/AIX)** tab.
19. Specify the **Language C++**.
20. Set the program to debug as **c:\Vaj3Ws\sampcode\hpj\P2hanoi.exe**.
21. Set the execution to **local**.

22. Click on *Load* and wait for the **Distributed Debugger** to open.
23. In the Source pane, go to **Line 68**: `System.out.println("Beginning puzzle. Solving for " + numberOfDisks + " disks.");`
24. Right-click in the line of code, select **Set Breakpoint** (or double-click on the line).
25. Click on the **Run** button, the program should now stop at line 68:
26. Mark the variable **numberOfDisks**, select **Add to Program Monitor** from the context menu.
27. Go to the expression monitor (bottom left pane), double-click the variable **numberOfDisks**, then change the value to **10**, press **Enter**.
28. Set another Breakpoint to **Line 70**: `puzzle.solve();` and to **Line 71**: `System.out.println("Puzzle solved!");`.
29. Click on the *Run* button, the program stops at Line 70.
30. Click *StepInto* button, to call the method solve.
31. Click the *StepOut* button.
32. The program is now stopped at line 71, go to the output window, and see the result of the program with 10 disks.
33. Select *Source -> Disassembly View* or *Source -> Mixed View* to see the machine code.
34. Click *Run* to let the program finish.

## Part IV. Debug a Interpreted Java Program

35. Go to a command line and start the debugger with the command **idebug** (or select *File -> Load program* in the debugger).
36. In the Load Program Dialog, go to the **Interpreted (Intel,AIX)** tab.
37. Click *Browse* for the class. **Add Directory** and find `c:\Vaj3Ws\sampcode\hpj`.
38. Select the directory and *Expand* the directory in the bottom pane until you see the `Hanoi.class`. Select the **Hanoi** class. Click *OK*.
39. Select **local execution** of the program. Click on *Load*.
40. If you want, you can repeat the debugging steps from Part III.

### What you did in this lab

- ☐ Create a Windows executable from inside VA Java and from the command line using the high-performance compiler
- ☐ Debug compiled programs and Java programs
- ☐ Setting breakpoints
- ☐ Step into methods
- ☐ Watch and change variable values during runtime





# E7 Notes Access Builder

## What this Exercise is About

In this lab we want to connect and retrieve information from a Notes database.

## What You Should Be Able To Do

At the end of this lab you should be able to

- ☐ Connect to a Notes database
- ☐ Retrieve information from a Notes database
- ☐ Use the Notes beans
- ☐ Generate JavaBeans using Domino Access Builder
- ☐ Use generated Domino Access Builder beans

## Introduction

In this example we want to learn how to access a Notes system from Java and how to build Java applications that access a Notes system.

The exercise can be done using a local Notes 5.0 client, or through an IIOP connection to a Notes/Domino server.

---

## Exercise Instructions

We start with preparing the necessary environment to run our application.

- ☐ If you have the WebSphere Test Environment or RMI IIOP project in your workspace, remove it (there is a conflict with packages loaded by Domino)
- ☐ Add the **Domino Access Builder Libraries** feature (*File-> Quick Start -> Features ->Add Feature*). This also loads the **Lotus Domino Java library 5.0**.
- ☐ For a local Notes client, the d:\Lotus\Notes directory must be in the system path.

These examples access the **Journal** and **AddressBook** databases of your Notes system. Also a **Test view** must be defined in the Journal database (you can easily create such a view in this database (*Create -> View*, then select all documents). Add few entries to the Journal database.

The easiest way to run the examples is by using a local Notes Client (Version 5). To run the examples against a Domino server using IIOP, the server has to turn on **IIOP** and **HTTP** services.

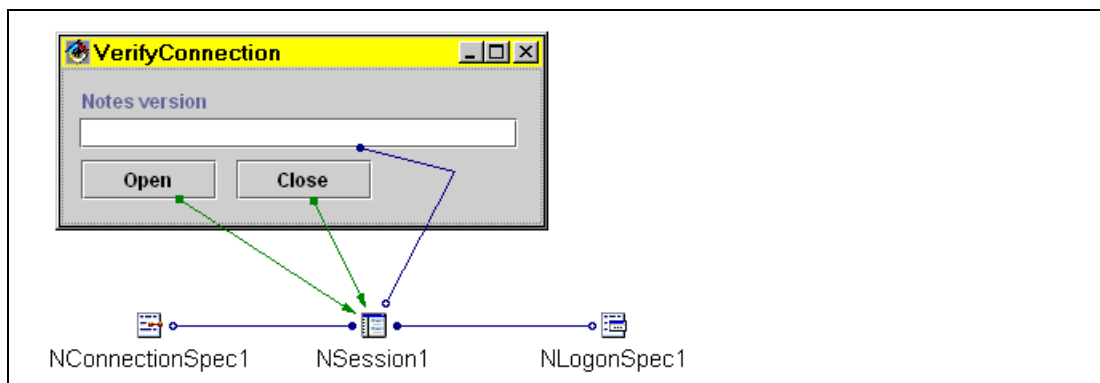
## Part I: Verify the Connection

1. Be sure to terminate any local **Notes client** (Version 5).
2. Open a scrapbook window and load the **sampcode\nab\VerifyConnection.scrap** file. Highlight the code to verify the local connection, and execute:

```
com.ibm.ivj.domino.ab.model.NConnectionSpec cs =
    com.ibm.ivj.domino.ab.model.NConnectionSpecFactory.newForLocalNotesClient();
cs.setDominoServerName("");
com.ibm.ivj.domino.ab.model.NSession s = new com.ibm.ivj.domino.ab.model.NSession();
s.setConnectionSpec(cs);
s.open();
s.close();
```

Your installation is fine if this code runs successfully.

3. For the rest of the exercises we import the code instead of developing it. Select the workshop project and import all the Java classes from the **sampcode\nab\java** directory; this creates the **itso.vaj3.nab** package.
4. Open the **ConnectingToNotes** class:

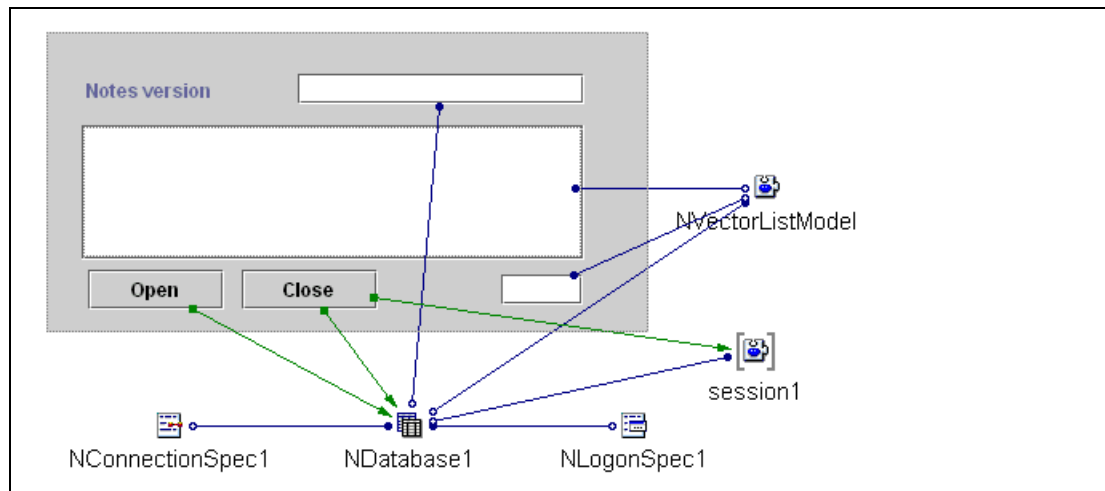


5. The class is setup to connect to a local Notes client, but you can also edit the **NConnectionSpec** and **NLogonSpec** beans to connect to a Domino server. Check the class path (*Bean -> Run -> Check Class Path*) to verify that the two Domino projects are included.
6. The *Close* button connects to the **closeOpenSessions** method. For IIOP this should be the close method.
7. Run the sample. After you click *Open* you should see the Notes version information, for example, *Release 5.0.1 (Intl) / 16 July 1999*.

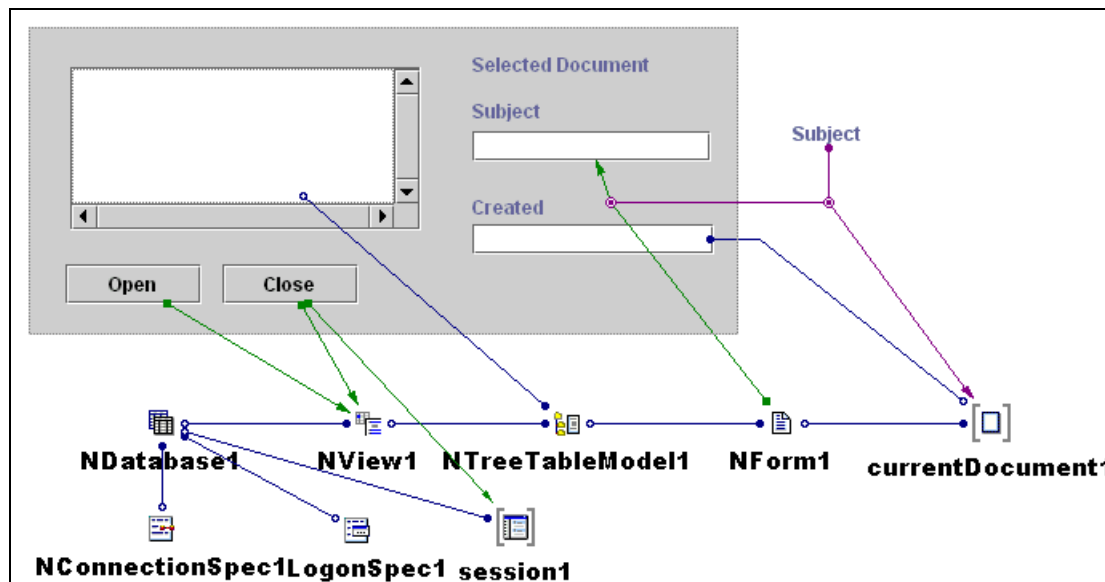
## Part II: Journal Database

The imported package contains a few examples: **ConnectingToNotes**, **JournalDbVector**, **JournalView**, **JournalViewBeans**, **NamesVector**, and **NamesContactsViewBeans**. We will describe the general steps to understand these examples by looking at the **JournalDbVector** and **NamesContactsViewBeans** classes.

1. Open the **JournalDbVector** class in the visual composition. This example is so simple that we will not describe it in detail. Basically the database (**Journal.nsf**) is opened and the collection of documents is listed in a JList. Notes provides the **NVectorListModel** for easy display of the collection in a GUI.



2. Check the class path (*Bean -> Run -> Check Class Path*), then run the sample. Then close the class.
3. Open the **JournalView** class in the visual composition. This example uses standard Notes beans only.



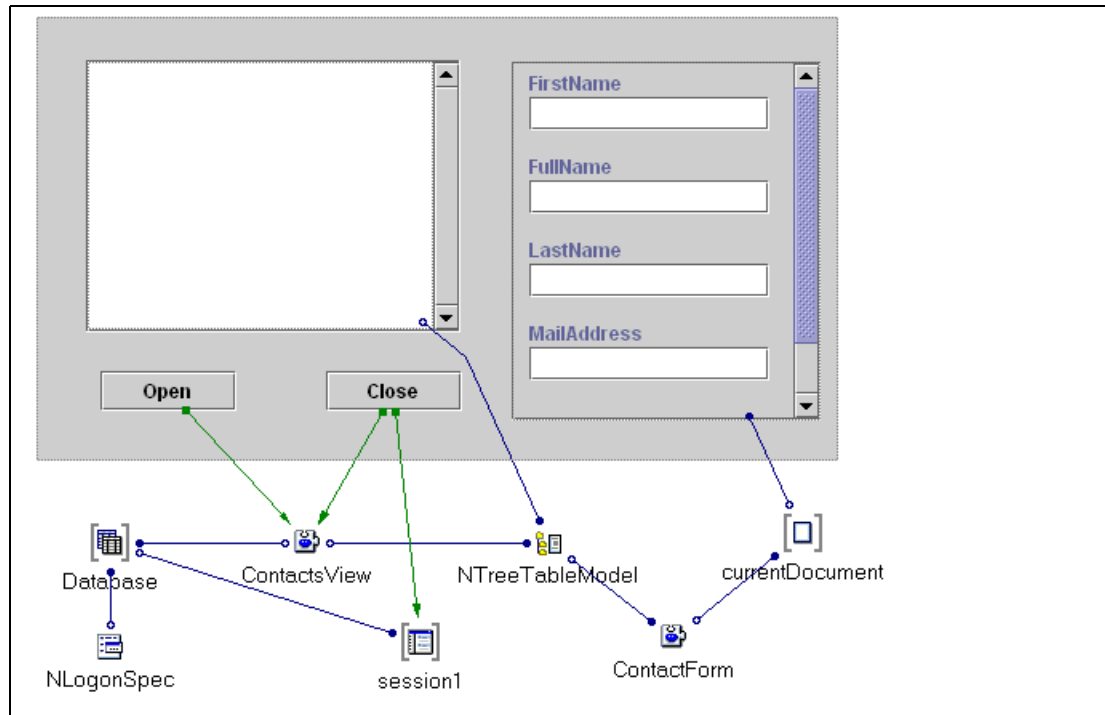
4. The GUI used a **NTreeTable** bean.

5. The non-visual beans and connections are:
  - A **NDatabase** bean and set **name** property to the **Journal.nsf** database.
  - A **NLogonSpec** bean and a **NConnectionSpec** bean, both connected to the database bean. For a local connection set **useLocal** to true. For an IIOP connection set **hostName** to the TCP/IP server name (xxxx.domain.ibm.com) and set appropriate userid and password.
  - A **NView** bean. Set the **name** property to **Test** (the name of a view in the database). Connect the **NView.database** property to the **NDatabase.this** property. Connect the Open and Close buttons to the methods of the NView bean.
  - A **NTreeTableModel** bean. Connect the **NTreeTableModel.treeTable** property to the GUI **NTreeTable.this** property. Connect the **NView.this** property to **NTreeTableModel.sourceView** property.
  - A **NForm** bean. Connect the **NtreeTableModel.currentDocument** property to the **NForm.currentDocument** property.
  - Tear-off the **currentDocument** property from the **NForm** bean. Connect the **NDocument.createDateTime** property to the **Created** text field (text). Connect the **NForm.currentDocument** event to the **Subject** text field and the connection parameter to **NDocument.getItemValue** method with **Subject** as parameter. (You can set the parameter as a constant or use a JLabel as in the illustration.)
  - Tear-off the **session** property from the database bean, and connect the **Close** button to the **closeOpenSessions** method of the session bean.
6. Check the class path (*Bean -> Run -> Check Class Path*), then run the sample.

## Part III. Generate an Access Bean Using Domino Access Builder

In this part we describe the **NamesContactsViewBeans** example.

7. First we have to generate domino access beans using Domino Access Builder features. Select the **itso.vaj3.nab** package and *Tools -> Domino Access Builder -> Properties*. Connect type is **Local**, then click *Finish*.  
For Domino server access, select Connect type as **Remote, IIOP**. Set **server name** to the TCP/IP name of the server, and set user and password specification to **Prompt when needed**.
8. Now select the **itso.vaj3.nab** package and *Tools -> Domino Access Builder -> Run*. You are prompted for the database name, using the **Browser** button log in the server and select the address database (**names.nsf**).
9. When you click *Next*, the new windows shows the **forms** in the database. Select **Contact**. You can specified the **fields to include** in the generated beans using the *Edit* option. Select **FirstName, LastName, FullName, MailAddress**.
10. Click *Next* and select a view (**Contact**) and the **columns** (*Edit* button) you want to see in the beans. In the final window you will see **an overview** of your selections. Click *Finish* to generate the beans.
11. Check in the Workspace what beans were generated.
12. Open the **NamesContactsViewBeans** class in visual composition (see below).
13. The visual parts are:
  - a **NTreeTable**
  - a **ContactFormQuickForm** that was generated by the Notes Access Builder
  - Open and Close buttons



14. The non-visual parts are:

- a **ContactView**, generated, contains the connection specification that was generated (**NamesNSFConnection**)
- a **ContactForm**, generated, contains the tailored form that includes the ContactDocument (also generated).
- Tear-off **currentDocument** from ContactForm
- the other beans are standard Notes beans (NDatabase, NLogonSpec, and NTreeTableModel)

15. Connections:

- ContactView, database to Database, this
- Database, logonSpec to NLogonSpec, this
- ContactView, this to NTreeTableModel, sourceView
- NTreeTableModel, currentDocument to ContactForm, currentDocument
- currentDocument, this to ContactFormQuickForm, document
- Tear-off the **session** property from the database bean, and connect the **Close** button to the **closeOpenSessions** method of the session bean.

16. Check the class path, and run the sample.

## Part IV. Deployment

If you **deploy** these applications, you must have the JAR files available in the class path:

- ☐ To access a Notes client, add **ablib30.jar** (IBMVJava\eam\runtime30\domino\ab) and **Notes.jar** (Lotus\Notes) to the class path
- ☐ To access a Notes server directly, add **ablib30.jar** (IBMVJava\eam\runtime30\domino\ab) and **NCSO.jar** (Lotus\Notes\data\domino\java) to the class path

When finished with the exercise, remove the **Domino Access Builder Libraries** and the **Lotus Domino Java library 5.0** features from the Workbench.

### What you did in this lab

- ☐ Connected to notes database
- ☐ Retrieved information from notes databases using the Notes JavaBeans
- ☐ Generated Domino Access Beans using Domino Access Builder
- ☐ Used generated Domino Access Beans to build applications

# E8 DB2 Stored Procedure Builder

## What this Exercise is About

In this lab we want to build and use DB2 stored procedures.

## What You Should Be Able To Do

At the end of this lab you should be able to

- ☐ Build a stored procedure
- ☐ Run a stored procedure
- ☐ Use a stored procedure in VisualAge for Java

## Introduction

In this example we want to learn how to use the DB2 Stored Procedures Builder.

---

## Exercise Instructions

Make sure DB2 6.1 is started.

### Part I: Create a Stored Procedure

1. Select (or open) the workshop project. The DB2 Procedure Builder can only be started from a project context.
2. Start the **DB2 Procedure Builder** (*Selected -> Tools - DB2 Stored Procedure Builder*). Be patient... it takes some time!
3. Open the **Environment properties** (*File* menu). Set the SAMPLE database (jdbc:db2:sample) and the COM.ibm.db2.jdbc.app.DB2Driver. You may have to enter userid/password for connecting.

4. Existing stored procedures are displayed. You can use **Get source** in the context menu and retrieve the source code of stored procedures.
5. Click on the *Insert Java Procedure* icon (or use the *Selected* menu) to create a new stored procedure. In the first page of the SmartGuide, name the new procedure **BigSalary**.
6. On the **Pattern** page, select **Run a single query**, **Return a result set**, and **Generate an SQL Exception**.
7. On the **SQL Query** page, click on *Define SQL*. This opens an **SQLAssistant**.
  - Select the tables USERID.DEPARTMENT and USERID.EMPLOYEE.
  - On the **Join** page select DEPTNO in department with WORKDEPT in employee and click on *Join*.
  - On the **Condition** page, select the SALARY column, the **Is greater than** operator, and click on *Variable* under values. Enter :SALARY as the variable name.
  - On the **Columns** page, select EMPNO, LASTNAME, SALARY from employee, and DEPTNAME from department.
  - On the SQL page you can see the SQL statement. Click *Finish* to generate.
8. Back on the SQL Query page, delete the USERID. prefix from the SQL statement by editing.
9. On the **Parameters** page you can see the SALARY input parameter.
10. On the **Options** page enter **BigSalary** as the specific name, **itso.vaj3.spb** as package, select **Dynamic SQL** and **Generate and Build**. Click on *Finish* to generate the code and register the stored procedure in DB2. Be patient ...
11. Study the generated Java source code and the messages.
12. Click on the *Run* icon, enter 35000 as the SALARY value, and click *OK* to run. A few employees with higher salary should be displayed in the message area.
13. Open the **DB2 Control Center** (enter userid/password when prompted). Expand to the *SAMPLE database -> Application Objects -> Stored Procedures*, and find the new **BigSalary** stored procedure. Close the Control Center.
14. Click on *File -> Save Project*.
15. Go to the Workbench (or Project) window and find the **BigSalary** class in the **itso.vaj3.spb** package.

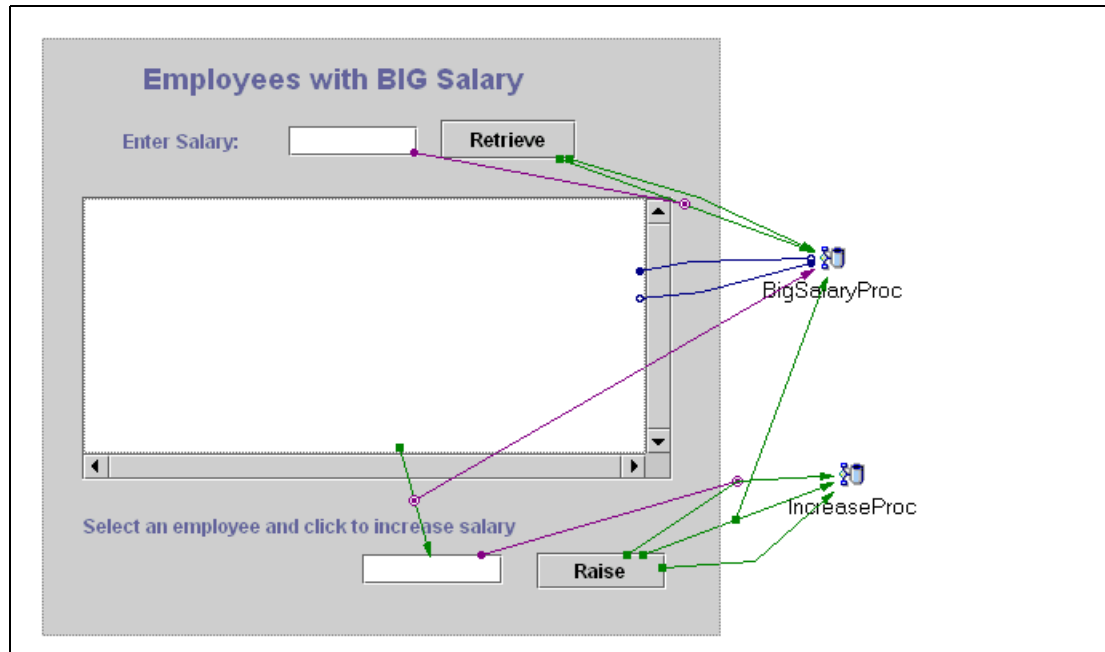
## Part II. Using a Stored Procedure

16. Import the two classes **EmployeeSalary** and **DbAccess** (.java files, from samPCODE\spb). The classes go into the same **itso.vaj3.spb** package.
17. Open the **EmployeeSalary** class in visual composition. This class invokes two stored procedures, **BigSalary** to retrieve employees, and **IncSalaryStp** to give one employee a raise.

You created BigSalary in Part I.

You may have created IncSalaryStp in the data bean exercise (Part VI); if not you cannot perform the increase function.





18. Study the application by opening the **BigSalary** ProcedureCall bean. We call this bean by setting the salary parameter and executing it. This displays the employees with higher salaries.
19. We can then select an employee and invoke the other stored procedure to increase the salary. We pass the employee number and a constant of 500 as the raise.
20. Run the application after checking the class path (data beans). Enter 30000 as the salary and find the employees. Select an employee and give a raise.

### What you did in this lab

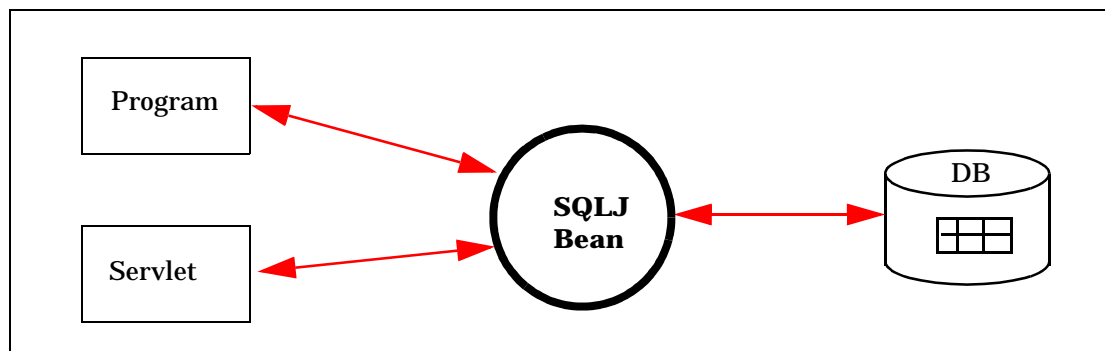
- ☐ Built a DB2 stored procedure
- ☐ Used a DB2 stored procedure
- ☐



# E9 Database Access with SQLJ

## What this Exercise is About

In this lab we want to access a DB2 table using SQLJ.



## What You Should Be Able To Do

At the end of this lab you should be able to

- ☐ Understand some SQLJ coding
- ☐ Translate an SQLJ program
- ☐ Run the DB2 Profiler
- ☐ Execute a program and a servlet that use the SQLJ program

## Introduction

We start with an existing SQLJ program, translate it, and then use it from a test program and from a servlet.

---

## Exercise Instructions

In the Workbench, load the **SQLJ Runtime Library** feature.

### Part I: An SQLJ Program

1. Study the code provided in the **sampcode\sqlj\SqljServBean.sqlj** file. The program retrieves the employees of a department and also counts the number of employees. Two select statements are issued.
2. Open the workshop project. Import the **sampcode\sqlj\SqljServBean.sqlj** file (*Workspace -> Tools -> SQLJ -> Import*). In the dialog deselect the **Perform translation** check box (we will do that later). Select your project and the SQLJ source file. Go to the Resources page of the project window and verify that the file is there.
3. Edit the file (*Resource -> Tools -> SQLJ -> Edit*). The VA Java provided editor opens. Close it. (Make sure that the .sqlj file is not read-only; otherwise translation fails.)
4. Define a tailored editor. In Windows -> Options, on the Resource Associations page, enter **.sqlj** as extension and click **Add**. Set up the external program **Notepad** (Winnt) for editing. Apply and close. Double click on the SQLJ file to edit.
5. Translate the SQLJ file (*Resource -> Tools -> SQLJ -> Translate*). The first time the **SQLJ Properties** window opens:
  - Leave encoding empty
  - Select *Perform online semantics checking*: COM.ibm.db2.jdbc.app.DB2Driver, jdbc:db2:sample URL, userid/passwordNote: you can setup the properties also in advance by selecting *Workspace -> Tools -> SQLJ -> Properties*.
6. A translation progress window opens, and a completed message appears. Check the Console window for messages.
7. The resulting code should be now in the **itso.vaj3.sqlj** package: SqljServBean, SqljServBean\_SJProfileKeys, DataCursor.
8. In the Resources window, refresh the view (*Window -> Refresh*) and you can see the generated .ser file in the **itso\vaj3\sqlj** subdirectory.

### Part II: Test Program

9. Import the **SqljTest.java** file from the **sampcode\sqlj** directory into your project. This is a small test program that allocates the SqljServ bean and executes its code.
10. To run this program the class path must be adjusted:
  - Try adding the **SQLJ Runtime Library** project to the class path and run the program.
  - If this does not work, remove the SQLJ Runtime Library from the project path and add the DB2 zip files to the workspace class path (*Window -> Options -> Resources*). Add **runtime.zip** and, if it exists, **sqlj.zip** (from SQLLIB\java).
11. The output of the program is in the Console window.

## Part III: Servlet

12. The `SqljServBean` can also be used from a servlet. Import the **SqljServlet.java** file from the **sampcode\sqlj** directory into your project. Study the **doGet** method.
13. To run the servlet, copy the **SqljDept.html** file (from `sampcode\sqlj`) to **d:\IBMJava\ide\project\_resources\IBM WebSphere Test Environment\hosts\default\_host\default\_app\web\Vaj3Ws** (you may have to create the **Vaj3Ws** subdirectory)
14. Make sure that the SQLJ Runtime library project is in the SERunner class path; then launch the WebSphere Test Environment (*Workspace -> Tools -> Launch ...*).
15. Run the servlet by entering **http://127.0.0.1:8080/Vaj3Ws/SqljDept.html** in the browser.

## Part IV. Profiler

16. Up to now we were using JDBC to run the DB2 access. If we have the **DB2 Software Developer's Kit** installed we can run the Profiler.
17. The translator created the profile **SqljServBean\_SJProfile0.ser** in `IBMJava\ide\project_resources\..workshop..\itso\vaj3\sqlj`. Copy the file to `Va3Ws\sampcode\sqlj` for easier access.
18. Run the **DB2 Profiler** against this file in a DOS Command window:

```
cd \Va3Ws\sampcode\sqlj
db2prof -url=jdbc:db2:sample SqljServBean_SJProfile0.ser
```
19. Rerun the two test programs.
20. In the **DB2 Control Center**, for the SAMPLE database, under *Application Objects -> Packages*, you can find an entry labeled: **SQLJSER0**.

When done, remove the **SQLJ Runtime Library** feature from the workbench.

### What you did in this lab

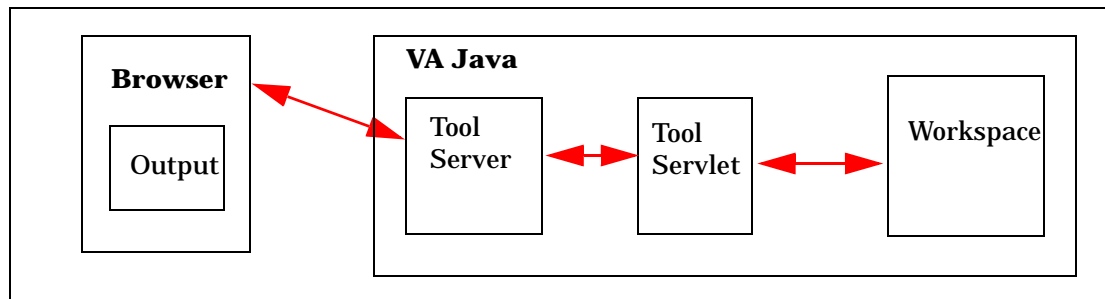
- ☐ Translated an SQLJ program using the SQLJ translator
- ☐ Tested the SQLJ access with a simple program and with a servlet
- ☐ Run the DB2 Profiler against this program



# E10 Remote Tool API and Tool Servlet

## What this Exercise is About

Running a tool servlet from a Browser.



## What You Should Be Able To Do

At the end of this lab you should be able to

- ☐ Understand the tool API
- ☐ Execute a remote tool

## Introduction

We prepare the system for the remote tool API and then run a small tool.

---

## Exercise Instructions

In the Workbench, check that the **IBM IDE Utility class libraries** project is loaded.

### Part I: Tool Servlet

1. Create a new project named **IBM Tool Server**.
2. Import the directory **c:\IBMJava\ide\tools\com-ibm-ivj-toolserver\com**, select the .class files.
3. Import the **ListProject.java** file from **sampcode\tool** into the workshop project. This creates a new **itso.vaj3.tool** package.
4. Study the code. This tool servlet is a subclass of the special **HttpServlet**. In the **doGet** method, the HTML output is generated. A **project** name is expected as parameter. The Tool API is used to connect to the workspace, access the project, and then get the packages of the project. For each package the types are retrieved. The HTML output lists the project, the packages, and the types of each package in an HTML table.
5. Find the **ToolHttpServer** class. Check the class path. You must include the **IBM IDE Utility class libraries** project and the workshop project. You also have to include the two directories: **c:\IBMJava\ide\project\_resources\IBM IDE Utility local implementation** and **...\IBM IDE UI class libraries**.
6. In the properties of the **ToolHttpServer** class, set the command line parameters as:  
-v -p 7777 -d .
7. Run the **ToolHttpServer** class. Check the Console for messages.
8. Start a browser and enter:  

```
http://127.0.0.1:7777/servlet/itso.vaj3.tool.ListProject?project=xxxx
    where xxxx=ITSO VA Java V3 Workshop    (or any other project name)
          xxxx=ITSO%20VA%20Java%20V3%20Workshop    (%20 for blanks in Netscape)
```
9. The output of the servlet should appear in the browser.

When done, remove the **IBM Tool Server** project from the workbench.

### What you did in this lab

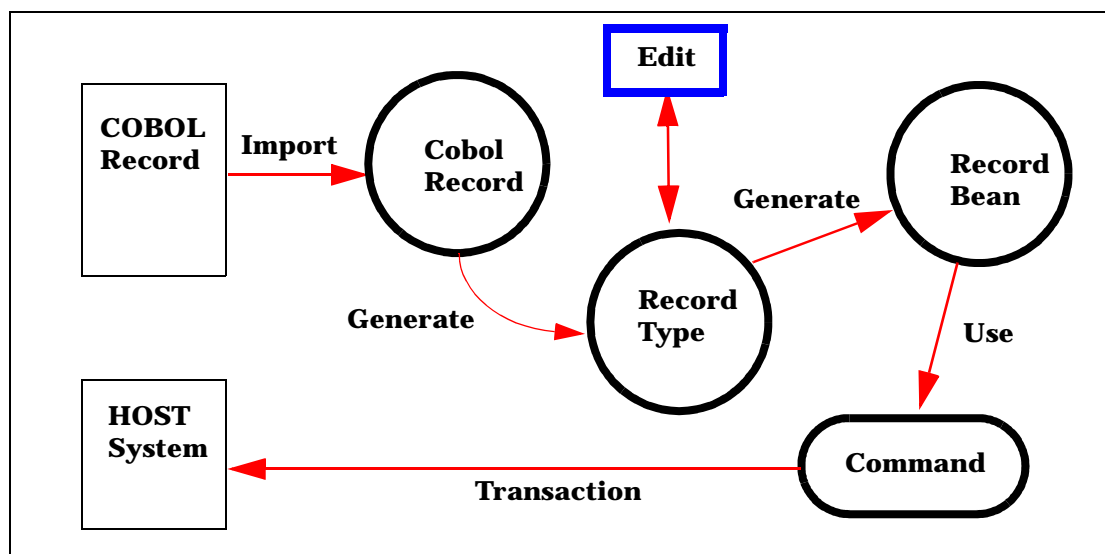
- ☐ Configured the Remote Tool Server and run a small tool servlet



# E11 Enterprise Access Builder for Transactions

## What this Exercise is About

In this lab we want to experiment with the record import and editing facilities. We also cover the construction of a command.



## What You Should Be Able To Do

At the end of this lab you should be able to

- ☐ Import records
- ☐ Work with records
- ☐ Understand commands, navigators, and mappers
- ☐ Understand connection and interaction specifications

## Introduction

We start with a COBOL record and progress towards a simple CICS transaction. We do not implement the real CICS interaction.

---

## Exercise Instructions

Add the **IBM Enterprise Access Builder Library** and the **CICS Connector** feature to the workbench.

### Part I: Import and Edit a COBOL Record

1. Create a new **itso.vaj3.eab** package in your workshop project.
1. Select *Workspace -> Tools ->Enterprise Access Builder -> Import COBOL to Record Type* to start the SmartGuide. Be patient ...
2. Select the **sampcode\eam\CDQuery.ccp** file and click *Next*.
3. Click on **View COBOL File** to see the content. Then select **all 01 areas** and click > to move them to the right pane, and click *Next*.
4. Check the workshop project and package (itso.vaj3.eab). Enter **CDQuery** as class name. Check *Continue working with the newly created record type*, and select **Edit record type**. Click *Finish*.
5. The **CDQuery** class is generated into the package, and the **Record Type Editor** is loaded.
6. Expand the records. Select the **SALES** field in the OUTLONGREC and change the dimension to **10** (you have to double-click on the value 12 in the IntArray Editor). Also change the **TEMP** field in the DHFCOMMAREA to a length of **140** (PICTURE clause).
7. Save the record (*Record -> Save*).
8. Create a record (*Record -> Create Record from the Record Type*). A new SmartGuide opens.
9. Check project and package names and enter **CDQueryRecord** as class name. Use *Direct access method, Dynamic Records, Generate with Notification*, and select **Use Inner Classes**. Click *Next*.
10. Leave all formats for NT. Click *Finish*.
11. At this point the actual code is generated. You will find the **CDQueryRecordType** and the **CDQueryRecord** classes in the Workbench.
12. The **Record Type Editor** is still open. Close those window.

## Part II. Creating a Command

13. Start the **Create Command SmartGuide** (*Tools ->Enterprise Access Builder -> Create Command*). Be patient ...
14. Select the workshop project and the **itso.vaj3.eab** package. Enter **CDCommand** as class name. Select *Edit when finished*. At this time we do not have any connection and interaction spec. Click *Next*.
15. Browse and select the **CDQueryRecord** as input record bean. We do not have a mapper class. Select **Use input bean as output**. Click *Finish*.
16. The **CDCommand** class is created and the **Command Editor** is started.
17. Select the **Input** bean, and then the **CDQueryRecord** in the right pane to see all the properties. Be patient ...
18. Add a connection spec for CICS: Select the **Connection** and *Tasks -> Add -> ConnectionSpec* and select the **CICSConnectionSpec**.
19. Select the resulting **CICSConnectionSpec** (right pane) to display the properties. In the bottom pane you would enter the CICS Server name and the URL (hostname).
20. Add an interaction spec for CICS: *Tasks -> Add -> InteractionSpec* and select the **ECIInteractionSpec**. Display its properties; this is where you would enter the CICS transaction and program names and userid/password to logon to the CICS system. When done, save the command.
21. The next step would be to build a **Navigators**, usually with Visual Composition. You would fill the input bean with the necessary values, then call the **execute** method of the Command, and then process the output record.
22. A **Mapper** could be used to automatically copy attributes from an existing bean to the input bean, and to copy attributes of the output bean to a business object. There is a SmartGuide to construct a mapper: *Tools ->Enterprise Access Builder -> Create Mapper*.

## Part III: Optional: Create an EAB Session EJB

Add the **EJB Development Environment** feature (this also loads the WebSphere Test Environment, etc).

23. On the **EJB** pane, create a new EJB group named **ITSO\_EAB\_EJBs**.
24. Select the new group and *Open to -> EAB Session Bean...* be patient. Click OK in the upcoming dialog to start the SmartGuide.
25. Select the workshop project and the **itso.vaj3.eab** package. Name the new class **CDQuerySession**. Select *Edit when finished*. Click *Browse* for the ConnectionSpec and find the **CICSConnectionSpec** class. Here we would click Edit to enter the CICS server name, etc. Click *Next*.
26. Click on *Add* for Method. Enter **run** as method name. Click on *Use existing command* and find the existing **CDCommand**. Click *Finish* to generate the EJB.
27. Wait for the EAB Session Editor. **Save** the bean and exit the editor.
28. Select the **run** method in the Members pane and *Add to ->EJB Remote Interface*. This makes the method available to the application.
29. Look at the **beforeInteraction** and **afterInteraction** methods. These would have to be tailored depending on the kind of interaction.

When done with the exercise, remove the **CICS Connector**, **HOD Connector**, and the **Enterprise Access Builder Library** features from the Workbench.

### **What you did in this lab**

- ☐ Imported an existing COBOL structure
- ☐ Edited the resulting COBOL record type
- ☐ Generate a record for an interaction with a host system
- ☐ Created a command to interact with a CICS system

## **Part 3**

# **Workshop Environment**



# A Setup and Installation

---

## Hardware Requirements

This is the hardware required to run the workshop:

- ☐ PCs with Pentium 300 MHz or better, 192 MB memory, 4 GB hard drive
  - Minimum: 233 MHz, 128 MB memory
- ☐ All PCs connected to a LAN (if possible)

---

## Software Requirements

This is the list of software that has to be pre-installed:

- ☐ **Windows NT Workstation** (or Server) **with SP 4** (or SP 5)
- ☐ **DB2 Version 6.1**
  - Select the **SDK** (Software Developer's Toolkit) during installation
  - Install **Fixpack 1a** or **Fixpack 2**
  - Create the SAMPLE database (run DB2SAMPL.EXE)
- ☐ Web browser: **Netscape Navigator 4.5** or higher, or **Internet Explorer 4** or higher
- ☐ **VisualAge for Java Version 3 Final Code**
  - Select custom install. Select all features except for ET/400 and ET/390.
  - Be sure to start the product after installation
  - Load the WebSphere Test Environment feature and save the workspace
  - Install the **Distributed Debugger**
- ☐ **Java Development Kit (JDK) 1.1.7** from IBM (or Sun)
  - Add c:\jdk1.1.7\bin to the PATH environment variable
- ☐ Optional: **Lotus Notes Client Version 5**

---

## Exercise Software

Unzip the provided file (va3ws.zip) to the C drive, this creates a **C:\Va3Ws** directory with the sample code.

---

## Exercise Database

The DB2 **SAMPLE** database must be created.

- ❑ The exercise instructions use userid/password to connect to the database. Define an NT user **USERID** with password **password**.

Open the **DB2 Control Center** and add the USERID user with all authorizations to the SAMPLE database.

- ❑ The solutions use USERID as schema name. The easiest way to achieve portability is to run these commands in a DB2 Command Window:

```
db2 connect to sample
db2 create alias userid.department for xxxxxx.department
db2 create alias userid.employee for xxxxxx.employee
    (where xxxxx is the normal NT logon ID)
db2 connect reset
```

- ❑ Configure DB2 to point to the JDK and to release stored procedures after use. In a DB2 Command Window enter:

```
db2 update dbm cfg using jdk11_path c:\jdk1.1.7
db2 update dbm cfg using keepdari no (not necessary with Fixpack 2)
```

You can verify the values in the **DB2 Control Center**. Select the DB2 instance and *Selected -> Configure*. You find the **Java Development Kit installation path** on the *Environment* page, and the **Keep DARI process indicator** on the *Applications* page. You must stop and restart DB2 for the changes to take effect.

```
db2stop
db2start
```



# **B** Special Notices

This publication is intended to help VisualAge for Java developers develop enterprise applications with VisualAge for Java Enterprise Version 3. The information in this publication is not intended as the specification of any programming interfaces that are provided by VisualAge for Java Enterprise. See the PUBLICATIONS section of the IBM Programming Announcement for VisualAge for Java Enterprise for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to

evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

You can reproduce a page in this document as a transparency, if that page has the copyright notice on it. The copyright notice must appear on each page being reproduced.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

IBM	AIX
AS/400	DB2
CICS	OS/2
OS/390	OS/400
S/390	SanFrancisco
TeamConnection	ThinkPad
VisualAge	WebSphere

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and/or other countries licensed exclusively through X/Open Company Limited.

SET and the SET logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.

# C Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

---

## International Technical Support Organization Publications

For information on ordering these ITSO publications see “How to Get ITSO Redbooks” on page 485.

- ❑ *VisualAge for Java Version 2: Persistence Builder with GUIs, Servlets, and Java Server Pages*, SG24-5426
- ❑ *IBM WebSphere and VisualAge for Java Database Integration with DB2, Oracle, and SQL Server*, SG24-5471
- ❑ *Developing an e-business Application for the IBM WebSphere Application Server*, SG24-5423
- ❑ *Enterprise JavaBeans Development Using VisualAge for Java*, SG24-5429
- ❑ *Using VisualAge Smalltalk ObjectExtender*, SG24-5258
- ❑ *VisualAge for Java Enterprise Version 2: Data Access Beans - Servlets - CICS Connector*, SG24-5265
- ❑ *Programming with VisualAge for Java Version 2*, SG24-5264, published by Prentice Hall, ISBN 0-13-021298-9, 1999 (IBM form number SR23-9016)
- ❑ *VisualAge for Java Enterprise Version 2 Team Support*, SG24-5245
- ❑ *Using VisualAge for Java Enterprise Version 2 to Develop CORBA and EJB Applications*, SG24-5276
- ❑ *VisualAge Java-RMI-Smalltalk, The ATM Sample from A to Z*, SG24-5418
- ❑ *Using VisualAge UML Designer*, SG24-4997
- ❑ *Application Development with VisualAge for Java Enterprise*, SG24-5081
- ❑ *Creating Java Applications with NetRexx*, SG24-2216
- ❑ *Unlimited Enterprise Access with Java and VisualAge Generator*, SG24-5246

---

## Redbooks on CD-ROMs

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at <http://www.redbooks.ibm.com/> for information about all the CD-ROMs offered, updates and formats.

CD-ROM Title	Collection Kit Number
System/390 Redbooks Collection	SK2T-2177
Networking and Systems Management Redbooks Collection	SK2T-6022
Transaction Processing and Data Management Redbooks Collection	SK2T-8038
Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
AS/400 Redbooks Collection	SK2T-2849
Netfinity Hardware and Software Redbooks Collection	SK2T-8046
RS/6000 Redbooks Collection (BkMgr)	SK2T-8040
RS/6000 Redbooks Collection (PDF Format)	SK2T-8043
Application Development Redbooks Collection	SK2T-8037
IBM Enterprise Storage and Systems Management Solutions	SK3T-3694

---

## Other Publications

These publications are also relevant as further information sources:

- ❑ *Developing JavaBeans with VisualAge for Java Version 2*, SC34-4735
- ❑ *Design Patterns: Elements of Reusable Object-Oriented Software*, Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, published by Addison-Wesley Professional Computing Series, ISBN 0-201-63361, 1995 (IBM form number SR28-5629)

---

## How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** <http://www.redbooks.ibm.com/>

Search for, view, download, or order hardcopy/CD-ROM redbooks from the redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this redbooks site.

Redpieces are redbooks in progress; not all redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

Send orders by e-mail including information from the redbooks fax order form to:

In United States	<b>e-mail address</b>
Outside North America	<a href="mailto:usib6fpl@ibmmail.com">usib6fpl@ibmmail.com</a>
	Contact information is in the "How to Order" section at this site:
	<a href="http://www.elink.ibm.link.ibm.com/pbl/pbl/">http://www.elink.ibm.link.ibm.com/pbl/pbl/</a>

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the "How to Order" section at this site:
	<a href="http://www.elink.ibm.link.ibm.com/pbl/pbl/">http://www.elink.ibm.link.ibm.com/pbl/pbl/</a>

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the "How to Order" section at this site:
	<a href="http://www.elink.ibm.link.ibm.com/pbl/pbl/">http://www.elink.ibm.link.ibm.com/pbl/pbl/</a>

This information was current at the time of publication, but is continually subject to change. The latest information may be found at the redbooks Web site.

### IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and redbooks by accessing the IBM Intranet Web site at <http://w3.itso.ibm.com/> and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may access MyNews at <http://w3.ibm.com/> for redbook, residency, and workshop announcements.

# IBM Redbook Fax Order Form

**Please send me the following:**

[illegible]

First name	Last name
------------	-----------

Company
---------

---

Address

City	Postal code	Country
------	-------------	---------

Telephone number	Telefax number	VAT number
------------------	----------------	------------

☐ Invoice to customer number \_\_\_\_\_☐ Credit card number \_\_\_\_\_

Credit card expiration date	Card issued to	Signature
-----------------------------	----------------	-----------

**We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.**

---

## List of Abbreviations

<b>API</b>	application programming interface
<b>ASP</b>	Active Server Pages
<b>ATM</b>	automated teller machine
<b>AWT</b>	Abstract Windowing Toolkit
<b>CGI</b>	Common Gateway Interface
<b>CORBA</b>	Common Object Request Broker Architecture
<b>DBMS</b>	database management system
<b>DLL</b>	dynamic link library
<b>GUI</b>	graphical user interface
<b>HTML</b>	Hypertext Markup Language
<b>HTTP</b>	Hypertext Transfer Protocol
<b>IBM</b>	International Business Machines Corporation
<b>IDE</b>	integrated development environment
<b>ITSO</b>	International Technical Support Organization
<b>JAR</b>	Java archive
<b>JDBC</b>	Java Database Connectivity
<b>JDK</b>	Java Developer's Kit
<b>JFC</b>	Java Foundation Classes
<b>JSBK</b>	Java Servlet Development Kit
<b>JSP</b>	Java Server Pages
<b>JVM</b>	Java Virtual Machine
<b>ODBC</b>	Open Database Connectivity
<b>PIN</b>	personal identification number
<b>RDBMS</b>	relational database management system
<b>RMI</b>	Remote Method Invocation
<b>SQL</b>	structured query language
<b>TCP/IP</b>	Transmission Control Protocol/Internet Protocol
<b>UOW</b>	unit of work
<b>URL</b>	uniform resource locator
<b>WWW</b>	World Wide Web





---

# ITSO Redbook Evaluation

ITSO Workshop: VisualAge for Java Version 3  
WS-SW-B402

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at <http://www.redbooks.ibm.com/>
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to [redbook@us.ibm.com](mailto:redbook@us.ibm.com)

Which of the following best describes you?

☐ **Customer**   ☐ **Business Partner**   ☐ **Solution Developer**   ☐ **IBM employee**  
☐ **None of the above**

**Please rate your overall satisfaction** with this book using the scale:  
**(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)**

Overall Satisfaction \_\_\_\_\_

**Please answer the following questions:**

Was this redbook published in time for your needs?      Yes\_\_\_ No\_\_\_

If no, please explain:

---

---

---

---

What other redbooks would you like to see published?

---

---

---

**Comments/Suggestions:      (THANK YOU FOR YOUR FEEDBACK!)**

---

---

---

---

---

**WS-SW-B402**  
**Printed in the U.S.A.**

